Using the Schur Product to Solve the Code Equivalence Problem

Michele Battagliola[1,2], Rocco Mora[3,4], and Paolo Santini[1]

[1] Università Politecnica delle Marche, Ancona, Italia
battagliola.michele@proton.me,
p.santini@univpm.it,
[2] University of St.Gallen, St. Gallen, Switzerland,
[3] CISPA-Helmholtz Center for Information Security, Sankt Ingbert, Germany
[4] University of Montpellier, Montpellier, France
rocco.mora@lirmm.fr

Abstract. Given two linear codes, the Code Equivalence Problem asks to find (if it exists) an isometry mapping one code into the other. A special case is the Permutation Equivalence Problem (PEP), where the isometry must be a permutation. The hardness of PEP is crucially dependent on the *hull* of a code, that is, the intersection between a code and its dual. Indeed, most of the known algorithms have running time that grows exponentially with the hull dimension. Since random codes have very small hull with large probability, PEP is deemed easy for random codes.

In this paper we study how the so-called Schur product between linear codes can be employed to solve PEP. The basic idea is to transform a given PEP instance by computing the square of the given codes. While it is well known that the square code operation preserves equivalence between linear codes, we show that, regardless of the hull dimension of the starting codes, their square codes have trivial hull with high probability. Furthermore, we show that as long as the code rate is sufficiently low, no additional permutations mapping the square codes exist with high probability. This effectively generates a new pair of equivalent codes with trivial hulls, where the underlying permutation remains identical to that of the original instance. This observation allows us to leverage existing hull-based attacks to recover the permutation for the square codes, and consequently, for the original codes. Furthermore, we improve this attack by exploiting the structural relationship between hulls: if a permutation maps two codes, the same permutation also maps their respective hulls. We show that by considering the square of the hull as a code in its own right, its hull also becomes trivial with high probability.

This allows for the identification of new weak instances of PEP, leading to an attack whose complexity no longer depends on the initial hull dimension, as it is the case of most known algorithm. In particular, we show that our attack achieves average polynomial-time complexity (since the square of the hull, when seen as a code, intersects with its dual in a low dimensional space with large probability) as long as $k < \sqrt{2n}$ or $h < \sqrt{2n}$, where n, k, and h denote the code length, dimension, and

hull dimension, respectively. We corroborate our analysis, which relies on some (plausible) heuristics, with intensive numerical simulations.
As a concrete application, we consider the updatable encryption scheme proposed by Albrecht, Benčina, and Lai at Eurocrypt 2025. All the recommended instances fall into the range of weak PEP instances identified in this paper; hence, they are susceptible to our attack. As a demonstration, we successfully recover the secret permutation for two of the instances claiming 128 bits of security in about 10 minutes on average on a laptop. As a fix, instances with hull dimension $h > \sqrt{2n}$ should be employed.

1 Introduction

Given two linear codes, the Code Equivalence Problem (in its decisional version) asks to decide whether there exists an *isometry* mapping one code into the other. An isometry can be seen as an isomorphism that preserves the weight distribution; from a geometric perspective, equivalent codes represent the same object and, de facto, an isometry can be thought of as a simple relabeling of the codewords. As with many other coding theory concepts, the problem has traditionally been studied in the context of the *Hamming metric*. Moreover, one typically looks only at the case in which the considered isometry is *linear*: the most general type of linear isometry corresponds to the group of monomial transformations, that is, permutations that also scale coordinates. In this case, the problem goes by the name of Linear Equivalence Problem (LEP). As special cases of LEP, we can consider the Permutation Equivalence Problem (PEP), where all the coefficients of the monomial are 1 (i.e., the isometry is a permutation), and the Signed Permutation Equivalence Problem (SPEP), where the coefficients of the monomial are either 1 or -1.[1]

Remarkably, the Code Equivalence Problem has also been studied in other contexts, e.g., the case of rank metric and matrix codes [CNP+23,RST24,CDAG20]. Interestingly, PEP can also be seen as a special case of the Lattice Isomorphism Problem (LIP) for modular lattices (permutation matrices are a special case of orthonormal transformations) [DvW22]. In a paper presented at Eurocrypt 2025, Albrecht, Benčina, and Lai have used PEP as the foundational security assumption for an updatable encryption scheme based on lattices [ABL25].

Hardness of PEP In 1997, Petrank and Roth showed that PEP is unlikely to be NP-complete (unless the polynomial hierarchy collapses) [PR97]. Moreover, in [SS13], Sendrier and Simos proposed a reduction which maps any LEP instance to a PEP instance. More recently, in [CSV25] Cheraghchi, Shagrithaya, and Veliche proposed polynomial-time reductions from PEP to both LEP and SPEP. These reductions take time which is polynomial in the finite field size.

[1] There exists also another definition of equivalence, called semi-linear LEP (Semi-LEP), in which one allows also for the application of a field automorphism.

It is immediate to see that the search version of PEP is at least as hard as the decisional version. Recently, Biasse and Micheli proposed a polynomial-time reduction for the other direction, i.e., from search PEP to decisional PEP [BM23].

In [PR97], Petrank and Roth showed that Graph Isomorphism Problem (GIP) reduces to PEP. It was widely conjectured that GIP is likely not NP-complete [Tor04,Mat79], as such a result would cause the polynomial-time hierarchy to collapse to its second level [BHZ87,Sch88]. A recent breakthrough by Babai [Bab16] shows that GIP can be solved in quasi-polynomial time. This algorithm makes the existence of efficient solvers for the code equivalence more likely (given the reductions and similarities between code equivalence and GIP [PR97,BOST19]). However, for many code equivalence instances, state-of-the-art solver are far from being efficient. Below, we provide a brief summary of known attacks.

Solvers for code equivalence The first algorithm to solve the Code Equivalence Problem was proposed by Leon in 1982 [Leo82]; in 2020, Beullens proposed a modification that, heuristically, improves the running time for big enough q [Beu20], with q being the finite field size. Even though these algorithms work for any pair of input codes, they require to first find low weight codewords, which is a notoriously hard problem in coding theory. So, they generically take time which is exponential in the code length. Another generic solver is based on finding collisions between canonical forms [CPS25,Now25]; using a meet-in-the-middle procedure, this takes time $O\left(\sqrt{\binom{n}{k}}\right)$, with n and k being the code length and dimension, respectively. Again, when k is linear in n, the cost of attacks based on canonical forms grows exponentially in the code length.

Notice that, modulo some minor technicalities, the aforementioned solvers can be used for both PEP and LEP. However, they do not provide efficient solvers, unless codes with extreme parameters (e.g., constant dimension) are considered. There is a significant gap between PEP and LEP: while the latter is deemed hard on average (given that the finite field is big enough), when it comes to PEP we know about some algorithm which, on average, turn into efficient solvers. Interestingly, as we recall next, the complexity of such solvers depends on the dimension of the hull, that is, the intersection between a linear code and its dual.

The Support Splitting Algorithm (SSA) [Sen00] is a solver for PEP, proposed by Sendrier in 2000. SSA is a general framework that allows to reconstruct the secret permutation by guessing its action on each index. As a fundamental subroutine, SSA requires a solver for decisional PEP, referred to as *signature function* in [Sen00]. In [Sen00], Sendrier proposes to instantiate a signature function from the weight enumerator of the hull: this results in an overall time for SSA which, heuristically, can be estimated as $O\left(n^3 + n^2 \log(n) q^h\right)$, with h being the hull dimension. In 2019, Bardet, Otmani and Saeed-Taha proposed a solver for the decisional version of PEP [BOST19]. They show that PEP for codes with trivial hull (i.e., with dimension 0) can be transformed, in polynomial time, into

a GIP instance in which the input graphs are undirected and weighted (the edges are labeled with finite field elements). Thanks to Babai's algorithm [Bab16], we know that GIP can be solved in quasi-polynomial time, so that all of these PEP instances can be solved in quasi-polynomial time as well in the worst case. In [BOST19], the authors show also how to perform the reduction to GIP when hulls are non-trivial; in this case, the reduction runs in worst case time $O\left(n^{h+\omega}\right)$, where ω is the exponent for matrix inversion.

In light of this discussion, PEP can generally be solved easily whenever the hull dimension h is small. Since for random codes the hull is very small with high probability [Sen97], both SSA and the reductions to GIP efficiently solve random instances of PEP. However, for all codes having a sufficiently large hull, these algorithms slow down and become inefficient. Since other known algorithms require finding low-weight codewords, instances with large hulls are deemed hard.

The square code The square code is a coding theory construction that extends the component-wise (Schur) product of two vectors to codes. In the last 15 years, it has turned out to be a powerful tool in the cryptanalysis of code-based cryptosystems, namely within the context of the McEliece scheme [McE78]. Here, security hinges, among other things, on the difficulty of distinguishing the public code from a random one or recovering the hidden structure.

For several McEliece variants, the square code has proven instrumental in structural attacks. In particular, these attacks exploit the fact that the square code of certain families of algebraic and structured codes has an unusually small dimension compared to that of a random linear code. Thus, by computing the square code of a public key, an attacker can detect non-random behavior and obtain a distinguishing attack. Sometimes, this is the first step toward a recovery attack, as knowledge of the low-dimension square code may lead to the partial or full reconstruction of the secret key. A non-exhaustive list of codes suffering from these techniques includes generalized Reed-Solomon (GRS) codes and their low-codimensional subcodes [Wie10], modified versions of GRS codes [CGGU$^+$14], Reed-Muller codes [BC14], Wild Goppa codes [COT16], algebraic geometry codes [CMCP17], polar codes [DBB18], subspace subcodes of RS codes [CL21], high-rate alternant/Goppa codes [FGUO$^+$13,MT23,BMT23], twisted GRS codes [CPTZ25].

A necessary condition for such a distinguisher to work is that the square code must not fill the ambient space, and this may impose conditions on the initial dimension of the target code. For instance, the binary Goppa codes used in the original proposal by McEliece or in the IND-CCA2 NIST candidate Classic McEliece [ABC$^+$20] resist this kind of attack, as their dimension is big enough to provide a trivial square code.

We will show that a similar constraint applies to our work, which instead focuses on the use of the square code for solving the code equivalence problem.

1.1 Our contribution

In this paper, we provide new insights and solvers for the Code Equivalence Problem, identifying several new weak instances for PEP.

We do this by stuyding how the Schur product can be employed as a powerful tool to improve known solvers for the Code Equivalence Problem. While the idea of exploiting the Schur product for PEP solvers was initially suggested by Saeed [Sae17, Appendix B], this observation has remained mainly of theoretical interest, as no practical algorithms leveraging this technique have been developed to date. More specifically, [Sae17] studies algebraic attacks and suggests enriching the algebraic system with additional linear equations that become available when the hull has a dimension at most the square root of the code length. This certainly has a positive impact on the resolution of the polynomial system. However, neither theoretical arguments nor numerical simulations were given to quantify the improvement obtained.

In this work, we start from the same observation in [Sae17] but follow a different path. First, we do not consider algebraic solvers for PEP. Moreover, we show that under the condition that the code rate is small enough, attacking directly the square codes is sufficient to efficiently solve the original PEP instance with high probability. As a demonstration of the novelty and efficacy of our approach, we show that our algorithm breaks the recommended instances of the updatable encryption scheme proposed by Albrecht, Benčina, and Lai at Eurocrypt 2025 [ABC+20]. As a matter of fact, we end up with an average polynomial-time attack for a family of weak instances which had not been attacked before.

The underlying idea is the following: instead of solving PEP on a pair of codes $(\mathscr{A}, \mathscr{B})$, we instead consider their powers $(\mathscr{A}^i, \mathscr{B}^i)$ for some integer $i \geqslant 2$. If $\mathscr{B} = \pi(\mathscr{A})$ for some permutation π, then one has also $\mathscr{B}^i = \pi(\mathscr{A}^i)$. Notice that the Schur product operation heavily modifies the code structure; for instance, it significantly increases the code dimension. For this reason, our approach can be employed only on codes having a sufficiently small dimension.

Computing powers of codes affects the hull: even if $\mathscr{A}$ and $\mathscr{B}$ have large hulls, this may not be true for $\mathscr{A}^i$ and $\mathscr{B}^i$. In this paper, we focus only on the case $i = 2$, i.e., on the so-called *squaring operation*. We focus only on the case of $\mathscr{A}$ and $\mathscr{B}$ being self-orthogonal codes, i.e., with hull dimension equal to the code dimension. This is not restrictive since, when the input codes have a large (but not full) hull, similarly to the SSA operating principle, we can just run our attack on the hulls (which, by definition, are self-orthogonal codes). We show that the hull dimension of the square of a self-orthogonal code has a somewhat different distribution with respect to that of random codes but, in any case, the hull is small with high probability. This implies that, with high probability, the PEP instance given by the square codes can be solved efficiently thanks to the reduction to GIP, as showed in [BOST19]. Notice that the claim about our algorithm being average polynomial-time algorithm is not in contrast with the fact that it reduces to the graph isomorphism problem and that Babai's algorithm takes quasi-polynomial time. Indeed, the latter solves worst-case instance, but here we only care about the average time complexity.

For such purposes, more efficient approaches are available and we rely on the solver described in Algorithm 2.

False positives and spurious solutions This simple approach we have described previously actually hides some subtle obstacles that need to be addressed. First, looking at the squared hull may lead to *false positives*. Indeed, it may happen that the squared hulls are permutation equivalent even if the initial codes are not. Analogously, when looking at the search version of PEP, one may find spurious solutions, i.e., permutations that solve PEP on the squared hulls but not on the initial codes. The existence of false positives and spurious solutions is somewhat intrinsic to the fact that the square of a self-orthogonal code has certain special properties. For instance, regardless of the dimension, the dual of the square code always contains the all-ones codeword. This means that the squared hull will never be a degenerate code but, if the hull dimension is too large, its square always corresponds to the dual of $\mathrm{Span}\big((1,\cdots,1)\big)$. For such a reason, looking at the squared hulls is not expected to work when $\frac{h(h+1)}{2} \geqslant n-1$.

More generally, whenever the squared hull has a too large dimension, say, $n-u$ for small u, false positives and spurious solutions are more likely to happen. Heuristically, we show that when $u > 1 + c\log_2(n)$ for some constant $c > 0$ that depends also on the finite field size, the impact of false positives becomes less meaningful. This condition should be interpreted as a rule of thumb, as it is obtained under the (heuristic) assumption that the square of the hull behaves as a random code. Nonetheless, this behavior is confirmed by numerical simulations.

Improving the algorithm with refinements Our basic algorithm can be improved by considering *refinements*, a concept that is somewhat similar to the techniques used in SSA. Starting from a given PEP instance, one can obtain several new PEP instances by applying code operations such as puncturing or shortening. These operations modify the code parameters and impact the hull as well. In this paper, we consider only the case of puncturing and show how it can be used to obtain more convenient parameters for the squared hull (namely, a larger dimension of the dual). We claim that puncturing in a small number of positions is sufficient to significantly reduce the impact of false positives. Numerical simulations corroborate this fact.

The algorithm with refinements is shown in Figure 1. Taking everything into account, our approach can be used to solve PEP instances with hull dimension $h < \sqrt{2n} - 0.5 = \sqrt{2n} \cdot \big(1 + o(1)\big)$. For these instances, on average, our algorithm runs in polynomial-time.

To confirm the effectiveness of our approach, we have implemented our algorithm and ran intensive numerical simulations on random weakly self-dual codes with various parameters. All the code we have used is publicly available `https://github.com/psantini1992/SquareCodeEquiv`. For a significant part of the paper, we focus on the case of low values of q, as the impact of false positives is expected to be more significant in such a regime. In cryptographic

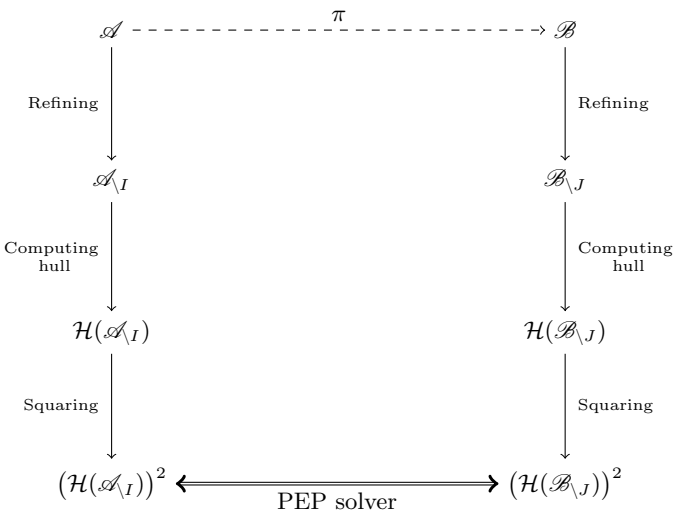


Fig. 1: Representation of how a PEP instance $(\mathscr{A}, \mathscr{B})$ can be transformed into a new PEP instance using the square code and refinements. In the figure, I and J are the sets used for the puncturing.

applications, the required values of q are typically much larger; in these cases, the impact of false positives should be extremely mild.

Cryptanalysis of the instances from [ABL25] As a practical application of our algorithm, we consider the code equivalence instances recommended in [ABL25]. Among other attacks, the hull dimension has been set so that both q^h and n^h (the times of SSA and the reduction to GIP, respectively) exceed 2^λ, with λ being the security parameter. However, the resulting instances have hull dimensions which are much less than $\sqrt{2n}$ and rather large values of q (e.g., $q \approx 2^{13}$): they constitute perfect use cases for our algorithm, which is expected to run in time $O(n^3)$ when GIP can be solved in average time at most $O(n^3)$ (we show that this is indeed the case). Notice that the instances from [ABL25] rely on SPEP; nonetheless, our algorithm can be applied to solve these instances as well. This is possible because a signed permutation mapping two codes also maps their duals; hence, one can simply focus on the hulls. Squaring the hull flattens the scalar coefficients (since the square of ± 1 is 1), effectively reducing the problem to a standard PEP instance. Once the permutation is found, retrieving the scalar coefficients boils down to solving an overdetermined linear system.

We have tested our Sage implementation on the two parameters sets for 128 bits of security from [ABL25]. We have considered 100 instances for each parameters set and have always been able to correctly retrieve the secret permutation, after an average time of dozens of minutes. Notice that the scheme from [ABL25]

can be easily made secure against our attack by choosing codes with larger hulls, say, a hull with dimension $h > \sqrt{2n}$.

1.2 Organization of the paper

In Section 2 we provide the notation used throughout the paper and the mathematical preliminaries. In Section 3 we describe and analyze our algorithm, based on using the Schur product to solve PEP. The analysis on the behaviour of the hull of the square of a self-orthogonal code is rather technical and is presented in Section 4. Finally, in Section 5 we describe the attack on the instances from [ABL25].

2 Preliminaries

In this section, we establish the notation used throughout the paper and recall essential background notions and facts regarding linear codes and the Code Equivalence Problem.

2.1 Notation

Matrices and vectors Throughout the paper, q will denote a prime power, i.e., $q = p^m$ for a prime p and a positive integer m. We will use $\mathbb{F}_q$ to indicate the finite field with q elements and $\mathbb{F}_q^*$ to indicate its multiplicative group. Bold lowercase (resp., uppercase) letters will be used for vectors (resp., matrices). The general linear group formed by the non-singular $k \times k$ matrices over $\mathbb{F}_q$ is denoted by $\mathrm{GL}_k(\mathbb{F}_q)$. The identity matrix of size k is indicated as $\mathbf{I}_k$, while $\mathbf{0}$ denotes the null matrix (its dimensions will always be clear from the context). Sometimes, we denote the set $\{1, \ldots, n\}$ as $[n]$. For a set $J \subseteq [n]$ and a vector $\mathbf{a} \in \mathbb{F}_q^n$, we denote $\mathbf{a}_J = (a_i)_{i \in J}$, i.e., $\mathbf{a}_J$ is the vector formed by the entries of $\mathbf{a}$ that are indexed by J. Analogously, $\mathbf{a}_{\setminus J} = \mathbf{a}_{[n] \setminus J} = (a_i)_{i \notin J}$.

Permutations We denote by S_n the symmetric group on n elements, and consider its elements as permutations on n objects. For a vector $\mathbf{a} = (a_1, \cdots, a_n)$ and $\pi \in S_n$, we have

$$\pi(\mathbf{a}) = \left(a_{\pi^{-1}(1)}, \cdots, a_{\pi^{-1}(n)}\right).$$

We naturally extend the action of permutations on matrices: $\pi(\mathbf{A})$ indicates the matrix resulting from the action of π on the columns of $\mathbf{A}$.

Sometimes we will represent a permutation π through the associated $n \times n$ matrix $\mathbf{P}$, so that $\pi(\mathbf{A}) = \mathbf{AP}$; with some abuse of notation, we write $\mathbf{P} \in S_n$ if $\mathbf{P}$ is an $n \times n$ permutation matrix.

2.2 Linear codes

A linear code $\mathscr{C} \subseteq \mathbb{F}_q^n$ with length n and dimension k, sometimes indicated as an $[n, k]$ code, is a k-dimensional subspace of $\mathbb{F}_q^n$. The quantity $R = k/n$ is called *code rate*, and any vector $\mathbf{c} \in \mathscr{C}$ is called *codeword*. A canonical representation for a code is through a *generator matrix*, that is, a full-rank matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ such that $\mathscr{C} = \left\{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_q^k\right\}$.

Given a code $\mathscr{C}$, its dual code $\mathscr{C}^\perp$ is the code comprising all vectors orthogonal to all the codewords of $\mathscr{C}$, namely: $\mathscr{C}^\perp = \left\{\mathbf{v} \in \mathbb{F}_q^n \mid \mathbf{cv}^\top = 0,\ \forall \mathbf{c} \in \mathscr{C}\right\}$. The dual code has dimension $n - k$ and is generated by a full-rank matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$, which is called *parity-check matrix*, for which $\mathbf{GH}^\top = \mathbf{0}$.

Definition 1 (Hull). *The* hull *of a code is defined as the intersection between the code and its dual:*

$$\mathcal{H}(\mathscr{C}) = \mathscr{C} \cap \mathscr{C}^\perp.$$

We say that the hull of a code is trivial if it contains only the null vector.

The hull is, by definition, a subcode of both $\mathscr{C}$ and $\mathscr{C}^\perp$, thus has dimension at most $\min\{k\ ,\ n-k\}$. As we will see in the next sections, the hull plays a crucial role in the complexity of many algorithms that solve PEP.

Definition 2. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a linear code with dimension k. We say that $\mathscr{C}$ is*

- *self-orthogonal if $\mathscr{C} \subseteq \mathscr{C}^\perp$*
- *self-dual if $\mathscr{C} = \mathscr{C}^\perp$*
- *weakly self-dual if $\mathcal{H}(\mathscr{C})$ is non trivial.*

2.3 Random codes and density of weakly-self dual codes

By random code, we mean a code sampled uniformly at random among the $\begin{bmatrix} n \\ k \end{bmatrix}_q$ codes having length n and dimension k, over $\mathbb{F}_q$. Here,

$$\begin{bmatrix} n \\ k \end{bmatrix}_q = \frac{(q^n-1)(q^{n-1}-1)\cdots(q^{n-k+1}-1)}{(q^k-1)(q^{k-1}-1)\cdots(q-1)} = \prod_{i=0}^{k-1} \frac{q^{n-i}-1}{q^{k-i}-1}$$

denotes the Gaussian binomial.

As one would expect, the probability that a random code has hull dimension h decreases as h grows and thus (weakly) self-dual codes are rather sparse. This matter has been thoroughly studied in [Sen97] and [SS13]; here, we recall one of the main results.

Theorem 1. *[SS13, Theorem 4] Let $\sigma_{n,k,h}$ indicate the number of weakly self-dual codes with hull dimension $h \geqslant 1$, over $\mathbb{F}_q$. When both n and k go to infinity, the ratio between $\sigma_{n,k,h}$ and $\begin{bmatrix} n \\ k \end{bmatrix}_q$ converges to ζ_h, where*

$$\zeta_h = \gamma_q \cdot \prod_{i=1}^{h} \frac{1}{q^i - 1},$$

where $\gamma_q = \prod_{i=1}^{\infty} q^i \cdot \frac{q^i-1}{q^{2i-1}}$. *In other words:*

$$\lim_{\substack{n\to\infty \\ k\to\infty}} \frac{\sigma_{n,k,h}}{\begin{bmatrix} n \\ k \end{bmatrix}_q} = \zeta_h.$$

Remark 1. *The quantity* ζ_h *can also be interpreted as the probability that a random* $[n,k]$ *code has hull dimension* h. *Notice that* ζ_h *does not depend on the code length nor on the code dimension, and decays as* q *and* h *grow. In particular, it decays exponentially with* h^2. *To see this, we start by approximating* γ_q *as* $1-1/q$ *and further continue as*

$$\begin{aligned}\zeta_h \approx (1-1/q)\cdot\prod_{i=1}^{h}\frac{1}{q^i-1} \approx \quad & (1-1/q)\cdot\textstyle\prod_{i=1}^{h} q^{-i} \\ & = (1-1/q)\cdot q^{-h(h+1)/2}.\end{aligned}$$

When q *is large, we further have* $1-1/q \approx 1$, *so we get a crude approximation of* ζ_h *as* $\zeta_h \approx q^{-h(h+1)/2}$. *For instance, for* $h=1$, *we get* $\zeta_1 \approx 1/q$, *while for* $h=2$ *we get* $\zeta_2 \approx 1/q^3$. *In particular, from [ABL25, Lemma 1], a random code has a trivial hull (i.e., with* $h=0$*) with probability*

$$\zeta_0 \geqslant 1 - 1/q - 1/q^2. \tag{1}$$

For large q, *this can be further approximated as* $1-1/q$.

2.4 Operations on codes

We recall some operations on code.

Definition 3 (**Puncturing**). *Given a linear code* $\mathscr{C} \subseteq \mathbb{F}_q^n$ *with dimension* k, *we define its puncturing in positions* $J \in [n]$ *as*

$$\mathcal{P}\big(\mathscr{C}, J\big) = \{(c_i)_{i\notin J} \mid (c_1,\cdots,c_n) \in \mathscr{C}\}.$$

For the sake of simplicity, sometimes we will write $\mathscr{C}_{\setminus J}$ in place of $\mathcal{P}(\mathscr{C}, J)$. It can be seen that $\mathscr{C}_{\setminus J}$ has dimension at most k and length $n - |J|$.

Definition 4 (**Shortening**). *Given a linear code* $\mathscr{C} \subseteq \mathbb{F}_q^n$ *with dimension* k, *we define its shortening in positions* $J \in [n]$ *as*

$$\mathcal{S}\big(\mathscr{C}, J\big) = \mathcal{P}\left(\mathscr{C}^*, J\right),$$

where

$$\mathscr{C}^* = \{\mathbf{c} \in \mathscr{C} \mid \mathbf{c}_J = (0,\cdots,0)\}.$$

For the sake of simplicity, sometimes we will write $\mathscr{C}_{-J}$ in place of $\mathcal{S}(\mathscr{C}, J)$. It can be seen that $\mathscr{C}_{-J}$ has dimension at most k and length $n - |J|$.

Proposition 1. *[HP10, Theorem 1.5.7] Let $\mathscr{C}$ be a linear code with length n and $J \subseteq [n]$. Then*

- $\mathcal{P}(\mathscr{C}, J)^{\perp} = \mathcal{S}(\mathscr{C}^{\perp}, J)$;
- $\mathcal{S}(\mathscr{C}, J)^{\perp} = \mathcal{P}(\mathscr{C}^{\perp}, J)$.

Given two vectors $\mathbf{a} = (a_1, \ldots, a_n), \mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{F}_q^n$, we define their *component-wise product*, denoted with the symbol $\star$, as follows:

$$\mathbf{a} \star \mathbf{b} := (a_1 b_1, \ldots, a_n, b_n).$$

This operation is also known as *Schur product* or *star product*. The component-wise product of vectors is clearly commutative and associative. We can thus use the notation $\bigstar_{\mathbf{a} \in S} \mathbf{a} = \mathbf{a}_1 \star (\mathbf{a}_2 \star (\cdots \star (\mathbf{a}_{s-1} \star \mathbf{a}_s)) \ldots)$ for a finite set $S = \{\mathbf{a}_1, \ldots, \mathbf{a}_s\}$, with the convention that $\bigstar_{\mathbf{a} \in \varnothing} \mathbf{a} = (1, \ldots, 1)$ and $\bigstar_{\mathbf{a} \in \{\mathbf{x}\}} \mathbf{a} = \mathbf{x}$. The notion of component-wise product can then be extended to linear spaces over finite fields, i.e. to linear codes.

Definition 5 (**Schur product of codes**). *Let $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ be two linear codes with length n, their Schur product $\mathscr{A} \star \mathscr{B}$ is defined as*

$$\mathscr{A} \star \mathscr{B} = \operatorname{span}(\{\mathbf{a} \star \mathbf{b} \mid \mathbf{a} \in \mathscr{A}, \mathbf{b} \in \mathscr{B}\}).$$

When the codes are the same, we speak of square code.

Definition 6 (**Square code**). *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a linear code with length n, the square code $\mathscr{C}^2$ of $\mathscr{C}$ is defined as*

$$\mathscr{C}^2 = \mathscr{C} \star \mathscr{C} = \operatorname{span}(\{\mathbf{c} \star \mathbf{c}' \mid \mathbf{c}, \mathbf{c}' \in \mathscr{C}\}).$$

One can define iteratively the Schur product among more than two codes. In particular, we have the *powers of a code*:

$$\mathscr{C}^{i+1} = \mathscr{C}^i \star \mathscr{C}.$$

To avoid confusion with the classical operation of squaring a matrix, if $\mathbf{G}$ is the generator matrix of $\mathscr{C}$, we denote with $\mathbf{G}^{[2]}$ the generator matrix of $\mathscr{C}^2$.

The following lemma on the inner product between component-wise products of vectors will also come in handy thereafter.

Lemma 1. *Let $S \subset \mathbb{F}_q^n$ be a set of vectors. Let $\{A, B\}$ and $\{C, D\}$ be two partitions of S. Then,*

$$\langle \bigstar_{\mathbf{a} \in A} \mathbf{a}, \bigstar_{\mathbf{b} \in B} \mathbf{b} \rangle = \langle \bigstar_{\mathbf{c} \in C} \mathbf{c}, \bigstar_{\mathbf{d} \in D} \mathbf{d} \rangle.$$

Proof. Both sides of the equation are equal to $\sum_{j \in [n]} \prod_{\mathbf{s} \in S} \mathbf{s}_j$. □

We now give some results on how squaring a (weakly)-self-dual code affects both its dual and its hull.

We start with the simple case of $q = 2$.

Theorem 2. *For any $\mathscr{C} \subseteq \mathbb{F}_2^n$, $\mathscr{C} \subseteq \mathscr{C}^2$ and $\mathscr{C}^{2\perp} \subseteq \mathscr{C}^{\perp}$.*

Proof. For any vector $\mathbf{c} \in \mathbb{F}_2$, it holds that $\mathbf{c} \star \mathbf{c} = \mathbf{c}$. So, $\mathscr{C}$ is a subcode of $\mathscr{C}^2$. By orthogonality, $\mathscr{C}^{2\perp} \subseteq \mathscr{C}^{\perp}$. □

For codes having a sufficiently large dimension (say, larger than $\sqrt{2n}$), it is likely that the square code gives the full vector space $\mathbb{F}_q^n$. However, for self-orthogonal codes, this is not true anymore, as the next theorem shows.

Theorem 3. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a self-orthogonal code. Then the dual of $\mathscr{C}^2$ is non-empty. In particular, $(1, \cdots, 1) \in (\mathscr{C}^2)^{\top}$.*

Proof. Since the code is self-orthogonal, for any two codewords $\mathbf{a}, \mathbf{b} \in \mathscr{C}$, it holds that

$$0 = \sum_{i=1}^{n} a_i b_i = (\mathbf{a} \star \mathbf{b}) \cdot (1, 1, \cdots, 1)^{\top}.$$

Since this holds for any pair of codewords $\mathbf{a}, \mathbf{b}$, then $(1, \cdots, 1)$ is orthogonal to all the codewords of $\mathscr{C}^2$. □

The above theorem implies that the square code of a self-orthogonal is never full. Moreover, it also tells us that two self-orthogonal codes with sufficiently large dimensions can have the same square code. This will become important in order to study the performances of the solvers we discuss later.

2.5 Permutation Code Equivalence Problem

For a code $\mathscr{C} \subseteq \mathbb{F}_q^n$ and $\pi \in S_n$, we denote

$$\pi(\mathscr{C}) = \{\pi(\mathbf{a}) \mid \mathbf{a} \in \mathscr{C}\}.$$

In other words, $\pi(\mathscr{C})$ is the code obtained by applying π to all codewords of $\mathscr{C}$. It is easy to see that $\mathscr{C}$ and $\pi(\mathscr{C})$ have some common properties, such as the same minimum distance and the same weight distribution. For this reason, we say that $\mathscr{C}$ and $\pi(\mathscr{C})$ are *permutation equivalent* and write $\mathscr{C} \equiv_{S_n} \pi(\mathscr{C})$.

Problem 1. ***Permutation Equivalence Problem (PEP)***
Given two linear codes $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$, decide if they are permutation equivalent, i.e., if there exists $\pi \in S_n$ such that $\mathscr{B} = \pi(\mathscr{A})$. In other words, given two generator matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}_q^{k \times n}$, decide if there exists an invertible matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ and a permutation matrix $\mathbf{P} \in S_n$ such that $\mathbf{B} = \mathbf{SAP}$.

In this paper we will sometimes focus on the search version of PEP, which asks to determine the permutation. According to the reduction in [BM23], the search and decisional versions are equivalent in terms of complexity.

Regarding equivalent codes and the Schur product, the following theorem holds.

Theorem 4. *[Ran15, Lemma 2.49] Let $\mathscr{B} = \pi(\mathscr{A})$ for some permutation $\pi \in S_n$. Then, $\mathscr{A}^2$ and $\mathscr{B}^2$ are linearly equivalent. In particular, $\mathscr{B}^2 = \pi(\mathscr{A}^2)$.*

Proof. Let us consider two codewords $\mathbf{a}, \mathbf{a}' \in \mathscr{A}$. Since $\mathscr{B} = \pi(\mathscr{A})$, there exist a pair of codewords $\mathbf{b}, \mathbf{b}' \in \mathscr{B}$ such that

$$\begin{aligned}\mathbf{b} &= (b_1, b_2, \cdots, b_n) \\ &= (a_{\pi^{-1}(1)}, a_{\pi^{-1}(2)}, \cdots, a_{\pi^{-1}(n)}),\end{aligned}$$

$$\begin{aligned}\mathbf{b}' &= (b'_1, b'_2, \cdots, b'_n) \\ &= (a'_{\pi^{-1}(1)}, a'_{\pi^{-1}(2)}, \cdots, a'_{\pi^{-1}(n)}).\end{aligned}$$

Let $\mathbf{c} = \mathbf{a} \star \mathbf{a}' \in \mathscr{A}^2$ and $\mathbf{c}' = \mathbf{b} \star \mathbf{b}' \in \mathscr{B}^2$. Observe that the j-th component of $\mathbf{c}$ is $a_j a'_j$, while the j-th component of $\mathbf{c}'$ is

$$b_j b'_j = a_{\pi^{-1}(j)} a'_{\pi^{-1}(j)} = c_{\pi^{-1}(j)}.$$

By linearity and considering all the components, we get $\mathscr{B}^2 \supseteq \pi(\mathscr{A}^2)$. The other inclusion is analogous. □

2.6 Solvers for code equivalence

We briefly summarize the most important solvers for PEP. While more efficient solvers for relaxed versions[2] exist, such as [BCDD+24,BN24], we only consider the standard case defined in Problem 1.

Algorithms based on low-weight codeword finding The first algorithm to solve the code equivalence was proposed by Leon in 1982 [Leo82]. The best generic solvers for this problem are ISD algorithms. Leon's algorithm requires to find, for both codes $\mathscr{A}$ and $\mathscr{B}$, all codewords having weight not greater than some w which

- cannot be too large, otherwise finding these codewords takes too much time, and
- cannot be too small, otherwise the two sets do not possess enough structure and reconstructing the map between the codes is not possible.

We are not aware of any precise criterion for choosing w properly; as a rule of thumb, many authors consider w to be slightly larger than the minimum distance d of the code. Given that finding low-weight codewords is a notoriously hard problem, this algorithm runs in exponential time. An improvement to Leon's algorithm has been proposed by Beullens in 2020 [Beu20]. This algorithm improves upon Leon's, only when q is large enough, since it requires only a fraction of weight-w codewords is necessary. This yields a polynomial speed-up with respect to Leon's algorithm, but, in any case, even Beullens' algorithm runs in exponential time.

[2] In such variants of PEP, the input comprehends multiple pairs of codes, all mapped by the same isometry.

The Support Splitting Algorithm Another algorithm to solve PEP is the Support Splitting Algorithm (SSA), proposed by Sendrier in 2000 [Sen00], which, in a nutshell, finds the unknown permutation by reconstructing its action index by index. This is obtained by using what Sendrier called *signature function*: a function s taking as input a code and an index $i \in \{1, \cdots, n\}$ that is invariant under permutations, that is,

$$s\big(\mathscr{A}, i\big) = s\big(\mathscr{B}, \pi(i)\big), \quad \forall i \in \{1, \cdots, n\}.$$

Given access to a *sufficiently discriminant* signature function (that is, a function such that $s(\mathscr{A}, i) \neq s(\mathscr{A}, j)$ for the wide majority of indices $i \neq j$), SSA can efficiently retrieve the secret permutation by progressively filtering out all pairs of indices that lead to different signatures. Eventually, a given signature function can be made more discriminant with the use of *refinements*.

Currently, the only known signature function requires computing the weight enumerator function of the hull, which contains q^h codewords. Considering polynomial overheads and the use of refinements, the algorithm has time complexity $O\left(n^3 + n^2 \log(n) q^h\right)$.

Reducing to Graph Isomorphism In 2019, Bardet, Otmani, and Saeed-Taha proposed a reduction from PEP to GIP taking worst-case time $O(n^{\omega+h})$, where ω is the exponent for matrix multiplication [BOST19]. When hulls are trivial, the reduction essentially requires computing some matrix inversions and multiplications. Because of Babai's algorithm [Bab16], we know that GIP can be solved in quasi-polynomial time, so that all PEP instances with trivial hulls can be solved in worst-case quasi-polynomial time as well. The reduction becomes slower when hulls are non-trivial. Still, for $h = O(1)$ it runs in polynomial time and, for $h = O\big(\log^a(n)\big)$, in quasi-polynomial time: in such cases, we have PEP can be solved in worst-case quasi-polynomial time.

We now recall the details of the reduction. For a code $\mathscr{A}$ with trivial hull and dimension k, any generator matrix $\mathbf{A}$ is such that $\mathrm{Rank}(\mathbf{A}\mathbf{A}^\top) = k$. Then, one can associate $\mathscr{A}$ with the following symmetric matrix

$$\mathbf{X} = \mathsf{Adj}(\mathbf{A}) = \mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^{-1} \mathbf{A} \in \mathbb{F}_q^{n \times n}.$$

The matrix $\mathbf{X}$ can be seen as the adjacency matrix for a simple, undirected and weighted graph with edge labels given by finite field elements. It is easy to see that the function is invariant under changes of basis, i.e., $\mathbf{X} = \mathsf{Adj}(\mathbf{A}) = \mathsf{Adj}(\mathbf{S}\mathbf{A})$ for any $\mathbf{S} \in GL_k(\mathbb{F}_q)$.

Now, let $\mathbf{B} = \mathbf{S}\mathbf{A}\mathbf{P}$ with $\mathbf{S} \in GL_k(\mathbb{F}_q)$ and $\mathbf{P} \in S_n$ be a generator matrix for $\mathscr{B}$; to the code $\mathscr{B}$, we associate a graph through the function Adj. Notice that

$$\begin{aligned}
\mathbf{Y} = \mathsf{Adj}(\mathbf{B}) &= \mathbf{B}^\top (\mathbf{B}\mathbf{B}^\top)^{-1} \mathbf{B} \\
&= \mathbf{P}^\top \mathbf{A}^\top \mathbf{S}^\top (\mathbf{S}\mathbf{A}\mathbf{A}^\top \mathbf{S}^\top)^{-1} \mathbf{S}\mathbf{A}\mathbf{P} \\
&= \mathbf{P}^\top \mathbf{A}^\top \mathbf{S}^\top \mathbf{S}^{-\top} (\mathbf{A}\mathbf{A}^\top)^{-1} \mathbf{S}^{-1} \mathbf{S}\mathbf{A}\mathbf{P} \\
&= \mathbf{P}^\top \underbrace{\mathbf{A}^\top (\mathbf{A}\mathbf{A}^\top)^{-1} \mathbf{A}}_{\mathbf{X}} \mathbf{P}.
\end{aligned}$$

Hence, the graphs whose adjacency matrices are $\mathbf{X}$ and $\mathbf{Y}$ are isomorphic and any permutation $\mathbf{P}$ so that $\mathbf{Y} = \mathbf{P}^\top \mathbf{X} \mathbf{P}$ gives also a solution to PEP.

If hulls are non-trivial, then the above transformation cannot be applied because $\mathbf{A}\mathbf{A}^\top$ (and $\mathbf{B}\mathbf{B}^\top$) does not have full rank. Yet, reducing to GIP is still possible, even though the reduction becomes more complicated. The authors in [BOST19] propose handling this case by shortening $\mathscr{A}$ and $\mathscr{B}$ on sets I and J, respectively, which must be information sets for the hulls: the resulting codes have trivial hulls and adjacency matrices can be computed. The reduction works when the set I is mapped into J; this requires enumerating, in the worst case, all candidates for J. For any pair I, J, the reduction requires other operations, such as puncturing the codes and enumerating all permutations that send I to J. All in all, the reduction takes time $O\left(n^{\omega+h}\right)$.

Remark 2. *Despite our work relies on a reduction to graph isomorphism, we do not use Babai's algorithm. Indeed, we are only interested in a probabilistic algorithm that succeeds with high probability. In this case, we rely a very simple test (Algorithm 2) that has polynomial complexity. This is discussed in the appendix.*

Algebraic solvers In [Sae17], the author discusses algebraic solvers for the Code Equivalence Problem. Notice that precise complexity estimates are not given and the author acknowledges that the proposed solver is efficient only when the code length is extremely small and the finite field size is small as well (say, $q \leqslant 5$). Remarkably, in the appendix of [Sae17], the authors comments about the possibility of using the squaring operation to gather new equations. In this paper, we are going to use the squaring operation as well but, as a major difference, do not consider algebraic solvers. Indeed, as we discuss next, when the hull dimension is small enough, the squares of the hulls are codes having trivial hull with large probability and this is enough to efficiently attack the problem.

3 Solving PEP with the Schur Product

In this section, we identify a wide range of new PEP instances that can be solved in polynomial time. As a basis for our solvers, we have the following fact:

Fact 1. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be an $[n, k]$ code with hull dimension h, sampled uniformly at random, and let $\mathscr{C}^2$ be its square code. If $k \leqslant \sqrt{2n}$, then $\mathcal{H}(\mathscr{C}^2)$ is small with overwhelming probability.*

The justification of this result is definitely non-trivial and, for this reason, we dedicate Section 4 to this matter. In this section, we describe how the above result can be used to efficiently solve certain PEP instances.

3.1 Basic idea

Let us recall Theorem 4: if $\mathscr{A}$ and $\mathscr{B}$ are two equivalent codes, then their squares $\mathscr{A}^2$ and $\mathscr{B}^2$ are equivalent as well. In particular, if $\mathscr{B} = \pi(\mathscr{A})$ for some permutation $\pi \in S_n$, then $\mathscr{B}^2 = \pi(\mathscr{A}^2)$.

In some cases, solving the equivalence problem on the pairs $(\mathscr{A}^2, \mathscr{B}^2)$ may be easier than solving it on the initial pair $(\mathscr{A}, \mathscr{B})$. Indeed, let us consider a PEP instance $(\mathscr{A}, \mathscr{B})$ in which both codes are self-orthogonal. Recalling Fact 1, after the squaring, the hulls of $\mathscr{A}$ and $\mathscr{B}$ are small with large probability. This unlocks the possibility of using an efficient PEP solver on the pair $(\mathscr{A}^2, \mathscr{B}^2)$.

The main limitation is that this attack can be applied only when the square codes do not fill the ambient space, i.e., when the dimensions of $\mathscr{A}^2$ and $\mathscr{B}^2$ are less than n. Indeed, if $\mathscr{A}^2 = \mathscr{B}^2 = \mathbb{F}_q^n$, then obviously the new instance is trivial, since any permutation would map a code into the other; in the following, we will call *degenerate* an $[n, n]$ code, i.e., a code that corresponds to the space $\mathbb{F}_q^n$.

As we show later on, to avoid squared codes that are degenerate, we require codes with dimension which is either $\leqslant \sqrt{2n}$ or $\geqslant n - \sqrt{2n}$ (since one can focus on the duals). Furthermore, we extend the attack to the case of codes of any dimension (even greater than $\sqrt{2n}$ or less than $n - \sqrt{2n}$) but such that their hull has dimension $h \leqslant \sqrt{2n}$. Essentially, we perform the attack by squaring only the hulls instead of the full codes.

False positives and spurious solutions As a minor drawback of our algorithm, we must deal with the potential existence of spurious solutions; that is, permutations that solve $(\mathscr{A}^2, \mathscr{B}^2)$ but not $(\mathscr{A}, \mathscr{B})$. The existence of such solutions could mislead our algorithm and, for this reason, we refer to them as "false positives". Notice that the existence of false positives may be due to two different phenomena:

- the codes $\mathscr{A}$ and $\mathscr{B}$ are not permutation equivalent but their hulls are equivalent, i.e.,
$$\mathscr{A} \not\equiv_{S_n} \mathscr{B} \quad \text{but} \quad \mathcal{H}(\mathscr{A}) \equiv_{S_n} \mathcal{H}(\mathscr{B}) ;$$
- the codes $\mathscr{A}$ and $\mathscr{B}$ are not permutation equivalent neither are their hulls, but the squares of the hulls are equivalent, i.e.,
$$\mathscr{A} \not\equiv_{S_n} \mathscr{B} \quad \text{and} \quad \mathcal{H}(\mathscr{A}) \not\equiv_{S_n} \mathcal{H}(\mathscr{B}) ,$$
but
$$\big(\mathcal{H}(\mathscr{A})\big)^2 \equiv_{S_n} \big(\mathcal{H}(\mathscr{B})\big)^2 .$$

More generally, even when the initial codes are permutation equivalent, there may exist spurious solutions, that is, solutions that would solve PEP on $\mathcal{H}(\mathscr{A})^2$ and $\mathcal{H}(\mathscr{B})^2$ but not PEP on $\mathscr{A}$ and $\mathscr{B}$.

Common sense suggests that such situations happen very rarely. We show that, with some caution for what concerns some technical caveats, this is indeed the case: thanks to heuristic arguments, we are able to show that the existence

of spurious solutions is extremely unlikely. Yet, when the squared hull has dimension $n-u$ with u too small, then false positives become rather likely and should be dealt with. To this end, we introduce the use of refinements.

Refining the algorithm Similar to the approach used in the SSA, we apply code operations (such as puncturing and shortening) to manipulate the parameters of the squared codes. Here, the goal is to reduce the impact of false positives and spurious solutions by increasing the codimension u of the squared hull. When u is too small, one can puncture the code on a set of size z to reduce the hull dimension up to $h-z$; this effectively increases the codimension of the squared hull. Based on heuristic arguments, we argue that setting $z = O\big(\log_2(n)\big)$ is sufficient in the worst case to make the occurrence of false positives and spurious solutions highly unlikely. In practice, through numerical simulations, we demonstrate that even setting z to a very small constant (e.g., $z = 1$) is often sufficient.

3.2 Non degenerate square codes

We first derive the conditions which guarantee that the square of a given $[n, k]$ code has dimension less than n. First, let $\mathbf{G}$ be the generator matrix for a code $\mathscr{C}$. Then, the generator matrix $\mathbf{G}^{[2]}$ of $\mathscr{C}^2$ satisfies the following properties:

Proposition 2. *If $\mathscr{C} \subseteq \mathbb{F}_q^n$ is a linear code with length n and dimension k, then $\mathscr{C}^2$ is a linear code with length n and dimension $k' \leqslant \min\left\{n\ ,\ \frac{k(k+1)}{2}\right\}$. More precisely, if $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ is a generator matrix for $\mathscr{C}$, then a generator matrix for $\mathscr{C}^2$ is a matrix $\mathbf{G}^{[2]}$ having as rows a maximal set of linearly independent vectors in the form*

$$\mathbf{g}_i \star \mathbf{g}_j$$

with $1 \leqslant i \leqslant j \leqslant k$, where $\mathbf{g}_i$ denotes the i-th row of $\mathbf{G}$.

Thus, to have that the dimension is less than n we ask that

$$\frac{k(k+1)}{2} < n,$$

which results in

$$k < k^* = \frac{-1+\sqrt{1+8n}}{2}.$$

It is easy to see that

$$\sqrt{2n} - \frac{1}{2} < k^* < \sqrt{2n}$$

hence $k^* = \sqrt{2n}\cdot\big(1+o(1)\big)$. So, whenever $k < k^*$, a code is guaranteed to have a square which is not degenerate, i.e., the dimension of the square code is strictly less than n.

We can actually extend the range of PEP instances which can be solved using our algorithm with the trivial observation that, if $\mathscr{A} \equiv_{S_n} \mathscr{B}$, then also $\mathscr{A}^\perp \equiv_{S_n} \mathscr{B}^\perp$. So, the very same attack can be performed by looking at the duals. In such a case, obtaining a non degenerate square code is guaranteed if

$$n - k < k^* \quad \Longrightarrow \quad k > n - k^*.$$

As we have already said, one can look only at the hulls. In such a case, the validity of our attack extends to all codes having any dimension k but hull of size $h < \sqrt{2n}$, that is, all weakly self-dual codes with hull of dimension $h < \sqrt{2n}$.

3.3 Non trivial automorphisms as a cause of false positives

Estimating precisely the probability with which false positives occur is definitely a hard task. However, with the help of some heuristics and combinatorial arguments, we are able to say something about this matter. Interestingly, we find that the dimensions of the squared hulls should not be too large, otherwise the probability that our algorithm gets fooled becomes larger.

As a warm up, let us consider two self-orthogonal codes $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ where $k = h$ is such that $\frac{h(h+1)}{2} = n-1$. For any pair $\mathscr{A}$, $\mathscr{B}$, we have $\mathscr{A}^2 = \mathscr{B}^2$. Indeed, recalling Theorem 3, we have $\mathscr{A}^{2\perp} = \mathscr{B}^{2\perp} = \mathrm{Span}\big((1, \cdots, 1)\big)$. In such a case, our algorithm will always end up concluding that the two codes are equivalent.

Hence, one should have at least $\frac{h(h+1)}{2} \leqslant n - 2$. Actually, as we argue next, issues may happen whenever $\frac{h(h+1)}{2} = n - u$ with u too small.

First, consider the following theorem.

Theorem 5. *Let $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ be two random codes with dimension $u - 1$ for some positive integer $1 < u < n$ and such that $(1, \cdots, 1) \in \mathbb{F}_q^n$ is contained in both $\mathscr{A}$ and $\mathscr{B}$. Then, the probability that $\mathscr{A} \equiv_{S_n} \mathscr{B}$ is not greater than $\frac{n!}{\begin{bmatrix} n-1 \\ u-1 \end{bmatrix}_q}$. In particular, for constant q and for $u = 1 + c\log_2(n)$ for some constant $c > \frac{1}{\log_2(q)}\left(1 - \frac{\log_2(e)}{\log_2(n)}\right)$, the probability decays exponentially with n.*

Proof. For any $\mathscr{A}$, the number of codes $\mathscr{B}$ which are permutation equivalent to $\mathscr{A}$ cannot be greater than $n!$. Since the number of choices for $\mathscr{B}$ is $\begin{bmatrix} n-1 \\ u-1 \end{bmatrix}_q$ [MS88, Theorem 4, p. 698], we obtain the probability in the thesis. We now study the asymptotic regime, assuming q is constant and n grows. To do this, we first find an upper bound on $\begin{bmatrix} n-1 \\ u-1 \end{bmatrix}_q = \prod_{i=0}^{u-2} \frac{q^{n-1-i}-1}{q^{u-1-i}-1}$. For any $x > y$, it holds that $\frac{x-1}{y-1} > \frac{x}{y}$; in our case, $x := q^{n-1-i}$ and $y := q^{u-1-i}$ and $x/y = q^{n-u} > 1$ by hypothesis. So, we get

$$\begin{bmatrix} n-1 \\ u-1 \end{bmatrix}_q \geqslant \prod_{i=0}^{u-2} \frac{q^{n-1-i}}{q^{u-1-i}} = q^{(u-1)(n-u)}.$$

Now, we use Stirling's expansion for the factorial and thus bound the probability as

$$\frac{n!}{\begin{bmatrix} n-1 \\ u-1 \end{bmatrix}_q} \leqslant \frac{n!}{q^{(u-1)(n-u)}} \sim 2^{n\log_2(n/e)-(u-1)(n-u)\log_2(q)}.$$

Setting $v = \log_2(n/e)$, we now take $u = 1 + c\log_2(n)$ for some constant $c > 0$, and further get

$$\begin{aligned} 2^{nv-(u-1)(n-u)\log_2(q)} &= 2^{n\cdot\left(v-(u-1)\log_2(q)(1-u/n)\right)} \\ &= 2^{n\cdot\left(v-c\log_2(q)\log_2(n)(1-u/n)\right)}. \end{aligned} \tag{2}$$

The term u/n grows as $\log_2(n)/n$ hence converges to 0 for growing n; thus, the exponent of the above probability asymptotically converges to

$$n \cdot \underbrace{\log\left(n^{1-c\log_2(q)} \cdot e^{-1}\right)}_{\zeta_c(n,q)}.$$

Whenever $\zeta_c(n,q)$ is negative, the probability in Eq. (2) decreases exponentially with n. By setting $\zeta_c(n,q) < 0$, we obtain

$$n^{1-c\log_2(q)} \cdot e^{-1} < 1 \quad \Longrightarrow \quad c > \frac{1}{\log_2(q)}\left(1 - \frac{\log_2(e)}{\log_2(n)}\right).$$

□

Remark 3. *Observe that the constant c is always less than 1. For instance, if $c = 2/\log_2(q)$, then the probability to sample two permutation equivalent codes decays at least as $2^{-n\cdot\log_2\left(e\cdot n\right)}$. In particular, we have that u must be at least $1 + 2\log_2(n)/\log_2(q)$.*

We now use the above theorem to estimate the probability that our algorithm encounters a false positive. We do this with the (strong) heuristic assumption that squared hulls behave as random codes.

Heuristic 1. *Let $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ be two random codes with hull dimension h, such that the squared hulls have dimension $n - u$. Then, if $u > 1 + \frac{\log_2(n)}{\log_2(q)}$, the probability to encounter false positives is negligible in n.*

This heuristic shall be used only as a rule of thumb to determine how large u should be. In practice, we shall use u as large as $1 + \log_2(n)/\log_2(q)$ to expect that the algorithm can succeed with large probability. For instance, as we are going to see in the next sections, whenever $u = 2$ and $n > q$, our algorithm fails with rather large probability. Indeed, $n > q$ implies $\log_2(n) > \log_2(q)$ so we shall have

$$u > 1 + \log_2(n)/\log_2(q) > 2.$$

3.4 Spurious solution for search PEP

Even when not encountering false positives, having squared hulls with a dimension that is too large may lead to issues when solving PEP. Assume $\mathscr{B} = \pi(\mathscr{A})$ for some permutation π and that $\mathcal{H}(\mathscr{B})^2$ has non-trivial permutation automorphisms. For each automorphism τ, we obtain a distinct solution to the PEP instance given by the squared hulls as $\pi \circ \tau^{-1}$. Obviously, the same applies to automorphisms of $\mathcal{H}(\mathscr{A})^2$.

Understanding how the automorphism group of the squared hulls behaves is a challenging task and is beyond the scope of this paper. Still, we are able to show that when u is too small, automorphisms must necessarily exist. This is established in the following theorem.

Theorem 6. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be a self-orthogonal code so that its square code has dimension $n - u$. If $u < 1 + \log_q(n)$, then the permutation automorphism group of $\mathscr{C}^2$ is non-trivial.*

Proof. Let $n - u$ indicate the dimension of $\mathscr{C}^2$, so that its dual $\mathscr{C}^{2\perp}$ has dimension u. Since the permutation automorphism group of a code corresponds to the permutation automorphism group of its dual, we can focus on $\mathscr{C}^{2\perp}$. From Theorem 3, we know that $(1, \cdots, 1) \in \mathscr{C}^{2\perp}$. Let us consider a generator for $\mathscr{C}^{2\perp}$, which will be in the form:

$$\begin{pmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{pmatrix},$$

where each $\mathbf{a}_i$ is a column vector of length $u - 1$. The maximum number of distinct vectors $\mathbf{a}_i$ we can have is q^{u-1}: whenever this value is less than n, then we have repeated entries. Any permutation that swaps repeated entries is an automorphism. □

Even when $\mathscr{C}^{2\perp}$ is large enough so that the above inequality is not satisfied, it may statistically happen that automorphisms given by identical coordinates exist.

Proposition 3. *Let $\mathbf{A} \in \mathbb{F}_q^{(u-1)\times n}$ be a uniformly random matrix. Then, the probability that $\mathbf{A}$ has all distinct columns is $\prod_{i=1}^{n-1}\left(1 - \frac{i}{q^{u-1}}\right)$. When $n \ll q^{u-1}$, this is well approximated by $e^{-\frac{1}{2}\cdot\frac{n^2}{q^{u-1}}}$.*

Proof. The probability simply corresponds to the probability that, when sampling uniformly at random n vectors from $\mathbb{F}_q$ from $\mathbb{F}_q^{u-1}$, they are all distinct. We now use the fact that, when $0 \leqslant x \ll 1$, it holds that $1 - x \approx e^{-x}$. Since $i < n < q^{u-1}$, then $1 - \frac{i}{q^{u-1}} \approx e^{\frac{-i}{q^{u-1}}}$; then

$$\prod_{i=1}^{n-1} 1 - \frac{i}{q^{u-1}} \approx e^{\frac{-\sum_{i=1}^{n-1} i}{q^{u-1}}} = e^{\frac{-n(n-1)}{2q^{u-1}}} \approx e^{\frac{-n^2}{2q^{u-1}}}.$$

□

Remark 4. *Setting $u = 1 + c\log_2(n)$ for some constant $c > 2/\log_2(q)$, we obtain that the above probability grows as $e^{-\frac{1}{2}n^{-a}}$ for some $a > 0$. Asymptotically, this is $1 - \frac{1}{2}n^{-a} = O(1)$.*

Observe that the previous analysis only tells us that u should be large enough to avoid that squared hulls have repeated coordinates. For what concerns existence of other automorphisms, we do not give any type of analysis. Still, heuristically, we expect that they exist with extremely small probability. Again, this intuition is confirmed by numerical simulations.

3.5 Using refinements

To reduce the impact of false positives and spurious solutions, one can apply some refinements on our algorithm. For instance, let $I, J \subseteq \{1, \cdots, n\}$ such that J is the image of I under the action of π, i.e., for any index $i \in I$ it holds that $\pi(i) \in J$. Then, one can set $\mathscr{A}_{\setminus I} = \mathcal{P}(\mathscr{A}, I)$ and $\mathscr{B}_{\setminus J} = \mathcal{P}(\mathscr{B}, J)$ and then use our attack on the pair $\left(\mathscr{A}_{\setminus I},\ \mathscr{B}_{\setminus J}\right)$. Whenever I and J are contained in at least one information set of the hulls, then the punctured codes will have hulls with dimension less than h. To this end, we consider the following theorem.

Theorem 7. *Let $\mathscr{A} \subseteq \mathbb{F}_q^n$ be a self-orthogonal code with dimension h. Let $I \subseteq \{1, \cdots, n\}$ with size z and such that there exists an information set containing I. Let $\mathscr{A}_{\setminus I} = \mathcal{P}(\mathscr{A}, I)$. Then, $\mathscr{A}_{\setminus I}$ has a hull with dimension $h - z$.*

Proof. For the sake of simplicity and without loss of generality, we give the proof for the case $z = 1$ and $I = \{1\}$, assuming I is contained in the information set $\{1, \cdots, h\}$. Let $\mathbf{G} \in \mathbb{F}_q^{h \times n}$ indicate the systematic generator matrix for $\mathscr{A}$. A generator for $\mathscr{A}_{\setminus I}$ is obtained by removing the first column of $\mathbf{G}$. Let $(g_{1,1}, g_{1,2}, \cdots, g_{1,n})$ indicate the first row of $\mathbf{G}$. Then, the first row of the generator matrix of the punctured code is $(g_{1,2}, \cdots, g_{1,n})$. Since $\mathscr{A}$ is self-orthogonal, we know that

$$0 = \sum_{j=1}^{n} g_{1,j}^2 = 1 + \sum_{j=2}^{n} g_{1,j}^2 \quad \Longrightarrow \quad -1 = \sum_{j=2}^{n} g_{1,j}^2.$$

Hence, $(g_{1,2}, \cdots, g_{1,n})$ and is not self-orthogonal and cannot be in the hull. For the other $h - 1$ rows, puncturing removes a 0 hence all properties regarding orthogonality are preserved. □

Our solver PEP using refinements is given in Algorithm 1. Basically, the algorithm performs a brute force over all possible punctured patterns for both codes. For each pair (I, J), the algorithm then solves search PEP on the punctured codes and, if it exists, finds a permutation $\tau \in S_{n-z}$. Then, one just needs to check whether this solution can be employed to obtain a solution for the initial instance $(\mathscr{A}, \mathscr{B})$. This can be done easily by working with generator matrices and employing trivial linear algebra.

Algorithm 1: Refined algorithm for solving PEP

Data: finite field size q, code length n, number z of punctured coordinates
Input: codes $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$
Output: permutation $\pi \in S_n$ such that $\pi(\mathscr{A}) = \mathscr{B}$, or failure

/* Puncture on sets of z coordinates and solve search PEP */
for $I \subseteq \{1, \cdots, n\}$, $|I| = z$ **do**
  Set $\mathscr{A}_{\setminus I} = \mathcal{P}(\mathscr{A}, I)$;
  for $J \subseteq \{1, \cdots, n\}$, $|J| = z$ **do**
    Set $\mathscr{B}_{\setminus J} = \mathcal{P}(\mathscr{B}, J)$;
    /* Solve search PEP on squared hulls of punctured codes */
    Compute $\left(\mathcal{H}(\mathscr{A}_{\setminus I})\right)^2$, $\left(\mathcal{H}(\mathscr{B}_{\setminus J})\right)^2$;
    If it exists, find $\tau \in S_{n-z}$ solving search PEP on $\left(\mathcal{H}(\mathscr{A}_{\setminus I})\right)^2$, $\left(\mathcal{H}(\mathscr{B}_{\setminus J})\right)^2$;
    /* See if τ leads to a solution for $\mathscr{A}$ and $\mathscr{B}$ */
    Compute $\mathbf{A} \in \mathbb{F}_q^{h \times (n-z)}$, generator matrix for $\mathscr{A}$;
    Compute $\mathbf{B} \in \mathbb{F}_q^{h \times (n-z)}$, generator matrix for $\mathscr{B}$;
    Find $\mathbf{S} \in GL_k(\mathbb{F}_q)$ such that $\mathbf{S} \cdot \tau(\mathbf{A}_{\setminus I}) = \mathbf{B}_{\setminus J}$;
    If it exists, find and return $\pi \in S_n$ such that $\pi(\mathbf{S} \cdot \mathbf{A}) = \mathbf{B}$;
/* No permutation has been found, codes are not equivalent */
return $\perp$

Relying on puncturing, one can increase the co-dimension of the squared hull. In particular, working with the hulls of $\mathscr{A}_{\setminus I}$ and $\mathscr{B}_{\setminus J}$ always leads to a dual having larger dimension. Indeed, let $n' = n - z$ and $h' = h - z$ denote the code length and hull dimension, respectively, after puncturing. Let us indicate with $u = n - \frac{h(h+1)}{2}$ the (maximum) co-dimension without puncturing. The co-dimension of the squared hull after puncturing is

$$\begin{aligned} n' - \frac{h'(h'+1)}{2} &= n - z - \frac{(h-z)(h-z+1)}{2} \\ &= \underbrace{n - \frac{h(h+1)}{2}}_{u} + \frac{2hz - z - z^2}{2}. \end{aligned}$$

This is always greater than the dimension one would have without puncturing (which is u), since $2hz - z - z^2 \geqslant 0$ for any $0 \leqslant z \leqslant 2h - 1$.

Example 1. *We make an explicit example of how refinements would work. Let us consider two weakly-self dual codes defined over $\mathbb{F}_{31}$, with length $n = 7$ and dimension $k = h = 3$. Let the generator matrices of these codes be defined as*

$$\mathbf{G} = \begin{pmatrix} 1\,0\,0\;\;3\;\;16\;25\;16 \\ 0\,1\,0\;26\;13\;18\;16 \\ 0\,0\,1\;16\;29\;18\;29 \end{pmatrix},$$

$$\mathbf{G}' = \begin{pmatrix} 14 & 19 & 25 & 22 & 5 & 17 & 2 \\ 0 & 27 & 22 & 14 & 22 & 29 & 5 \\ 16 & 16 & 19 & 27 & 5 & 10 & 3 \end{pmatrix}.$$

The permutation sending the first code onto the second code is

$$\pi = (2, 6, 5, 4, 7, 3, 1),$$

i.e., the permutation sending the second coordinate to the first position, the sixth coordinates to the second positions, etc. The squares of these codes have dimension $\frac{h(h+1)}{2} = 6 = n-1$ *but trivial hulls. Since we know that the all-ones codeword must be always be in the dual of a self-dual code, we know that*

$$\mathscr{A}^{2\perp} = \mathscr{B}^{2\perp} = \mathrm{Span}\big((1, \cdots, 1)\big).$$

Any permutation would be a solution to PEP defined on $(\mathscr{A}^2, \mathscr{B}^2)$*. We apply algorithm 1 using* $I = \{1\}$ *and* $J = \{7\}$ *(i.e., use* $z = 1$*). The hulls of the punctured codes* $\mathscr{A}_{\setminus I}$ *and* $\mathscr{B}_{\setminus J}$ *have dimension* 2 *and are generated by the following matrices*

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 26 & 13 & 18 & 16 \\ 0 & 1 & 16 & 29 & 18 & 29 \end{pmatrix},$$

$$\mathbf{U}' = \begin{pmatrix} 1 & 0 & 15 & 10 & 18 & 30 \\ 0 & 1 & 24 & 25 & 24 & 19 \end{pmatrix}.$$

We now consider the codes which are generated by the squares of $\mathbf{U}$ *and* $\mathbf{U}'$*: the codes obtained from these matrices have trivial hull and dimension* 3*. Relying on the reduction to GIP, we obtain the two following adjacency matrices:*

$$\mathbf{A} = \begin{pmatrix} 25 & 2 & 11 & 6 & 0 & 18 \\ 2 & 0 & 6 & 23 & 14 & 17 \\ 11 & 6 & 0 & 18 & 7 & 20 \\ 6 & 23 & 18 & 2 & 18 & 26 \\ 0 & 14 & 7 & 18 & 9 & 14 \\ 18 & 17 & 20 & 26 & 14 & 29 \end{pmatrix},$$

$$\mathbf{A}' = \begin{pmatrix} 25 & 0 & 6 & 11 & 18 & 2 \\ 0 & 9 & 18 & 7 & 14 & 14 \\ 6 & 18 & 2 & 18 & 26 & 23 \\ 11 & 7 & 18 & 0 & 20 & 6 \\ 18 & 14 & 26 & 20 & 29 & 17 \\ 2 & 14 & 23 & 6 & 17 & 0 \end{pmatrix}.$$

We find that the solution of GIP is $\tau = (1, 5, 4, 3, 6, 2)$*. With linear algebra, we discover that*

$$\mathbf{S} = \begin{pmatrix} 2 & 14 & 17 \\ 5 & 0 & 29 \\ 3 & 16 & 10 \end{pmatrix}$$

is the 3×3 *change of basis such that* $\mathbf{S} \cdot \tau(\mathbf{G}_{\setminus I}) = \mathbf{G}'_{\setminus J}$*. Now, we compute* $\mathbf{SG}$*:*

$$\mathbf{SG} = \begin{pmatrix} 2 & 14 & 17 & 22 & 25 & 19 & 5 \\ 5 & 0 & 29 & 14 & 22 & 27 & 22 \\ 3 & 16 & 10 & 27 & 19 & 16 & 5 \end{pmatrix}.$$

Matching the columns of $\mathbf{SG}$ *with those of* $\mathbf{G}'$*, we reconstruct* π*.*

3.6 Numerical simulations

We now show the effectiveness of our approach by reporting the results of numerical simulations. We tested our attack on random PEP instances given by weakly self-dual codes, for several code parameters n, q, and h.

As a solver for PEP, we have employed the reduction to GIP, relying on a very simple solver for GIP, which we detail and analyze in Appendix A. This algorithm reconstructs the permutation using the entries of the rows of adjacency matrices as an invariant. Namely, let $\mathbf{X}, \mathbf{Y} \in \mathbb{F}_q^{n \times n}$ be two adjacency matrices: if row i of $\mathbf{X}$ and row j of $\mathbf{Y}$ are the unique rows having the same entry multiset, then the permutation must send i to j. When a row matches with more than one row, the algorithm fails. Notice that, obviously, there are much better algorithms for GIP. Still, even this basic solver is powerful enough when the finite field is sufficiently large: for $q \geqslant 5$ and under the heuristic assumption that the adjacency matrices we consider behave as random $n \times n$ matrices over $\mathbb{F}_q$, it has constant success probability. In particular, the success probability approaches 1 as soon as $q \geqslant 7$ and n grows.[3] See Appendix A for details.

Remark 5. *We remark that more sophisticated algorithms and tools for solving GIP are available (e.g., [MP14]). However, deriving the expected running time of these algorithms, for the class of graphs we are interested in, is arguably not easy (and, perhaps, out of the scopes of this paper). Remember that, in this paper, our primary interest is in showing that certain PEP instances can be solved, with high probability, in average polynomial time. For these reasons, we chose to rely on a much simpler solver for GIP, running in polynomial time (quadratic in* n*) and succeeding with overwhelming probability when* q *is large enough (see Appendix A).*

In Figure 2, we report the results of numerical simulations regarding the success rate of our attack, considering random self-orthogonal codes defined over $\mathbb{F}_5$. In these simulations, we have $n = 30$ and several values of $k = h$. For all instances, we have considered two versions of our attack: the elementary one without refinements, and the one with refinements with $z = 1$ (i.e., puncturing on a single position). Observe that for a very small hull, our algorithm has a very

[3] This requires also assuming that the adjacency matrices we use in the reduction from PEP to GIP do not exhibit any "exotic" property. Indeed, these matrices are obtained with a very specific construction, starting from rectangular matrices which, in our case, are obtained by squaring $k \times n$ matrices.

low success probability. For such cases, the algorithm from [BOST19] should be employed. The success rate increases with growing h.

If we increase q, we expect that the success rate increases. This can indeed be seen in Figure 2, in which we have considered also $q = 7$ and $q = 61$. Going from $q = 5$ to $q = 7$ is already enough to significantly increase the success rate. Using $q = 61$, we see that even the basic version of our algorithm succeeds almost always, apart from the case $h = 7$. Using refinements, again, we obviously increase the success rates in all cases.

Observe that the codimension of the squared hull plays a crucial role in determining the success rate of our algorithm. For instance, for $h = 6$, we have $\frac{h(h+1)}{2} = 21$; hence, the minimum codimension of the squared hull is $n - \frac{h(h+1)}{2} = 9$. This is much greater than $1 + 2\log_2(n)/\log_2(q) \approx 3.9$, so we do not expect issues due to spurious solutions. Instead, for $h = 7$, we have $u = n - \frac{h(h+1)}{2} = 2$ and we see that, regardless of q, the basic algorithm always fails. Using refinements with $z = 1$, the algorithm has always correctly found the valid permutation. This is something we expected; indeed, for $z = 1$, the codimension is given by

$$n - z - \frac{(h-z)(h-z+1)}{2} = 29 - \frac{6 \cdot 7}{2} = 8,$$

which is much greater than $1 + 2\log_2(n)/\log_2(q)$.

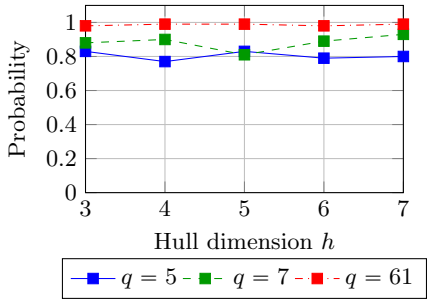
Observe that, for large q, the algorithm succeeds with probability almost 1, apart from the cases in which the codimension is very small.

Finally, in Figure 3 we report about experiments conducted with growing values of n and self-orthogonal codes with $h = k = \lfloor \alpha\sqrt{n} \rfloor$, with $0 < \alpha \leqslant \sqrt{2}$. We have first considered the case of $\alpha = 1$, i.e., $h = \lfloor n \rfloor$. In such a case the co-dimension of the squared hull is approximately

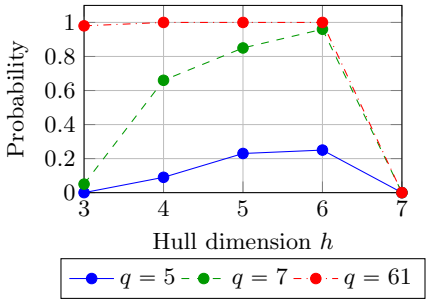
$$n - \frac{\sqrt{n}(1+\sqrt{n})}{2} \approx \frac{n}{2},$$

so we expect that false positives occur with very small probability. This is confirmed by numerical simulations (see Figure 3a). In any case, the success probability increases for growing n, apart from the case $q = 5$. In such a case, our solver for GIP is expected to have a constant success rate which is less than 1 (see Appendix A). Using refinements, the success rate increases significantly, even for $q = 5$ (see Figure 3b).

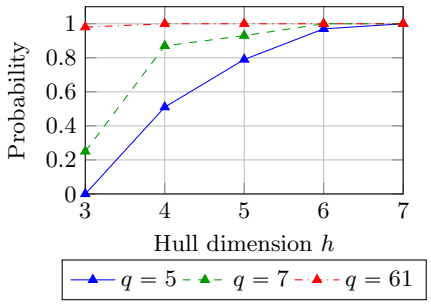
We have also considered the case of $h = \lfloor \sqrt{2n} - \frac{1}{2} \rfloor$, for which in many cases the codimension of the squared hull is very low. When this happens, the success probability of our algorithm drops (see Figure 3c). For instance, our basic version fails with large probability, regardless of q, for $n = 30$, for which $h = 7$ hence $u = n - \frac{h(h+1)}{2} = 2$. As the figure shows, the success probability becomes lower whenever the codimension of the squared hull is low. In any case, refinements can greatly boost the algorithm and raise the success probability, even for the case $q = 5$ (see Figure 3d).



(a) Probability of trivial hull



(b) Success rate, no refinements



(c) Success rate, with refinements

Fig. 2: Results of numerical simulation on self-orthogonal codes with $n = 30$, $k = h$ and different values of q.

4 The hull of the square of self-orthogonal codes

We discuss here the validity of Fact 1, which provide the theoretical basis for the attack to PEP for self-orthogonal codes.

Given two $[n, k]$ codes $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ and their respective generator matrices

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_k \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_k \end{pmatrix},$$

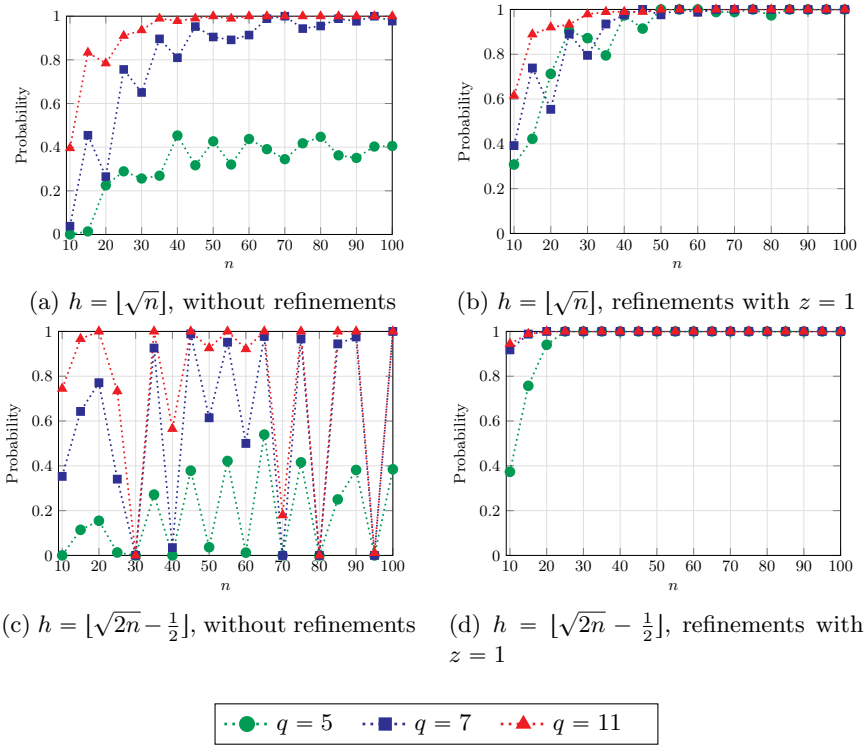
let us consider the $k \times k$ matrix

$$\mathbf{A}\mathbf{B}^\top = (\langle \mathbf{a}_i, \mathbf{b}_j \rangle)_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant k}} \tag{3}$$

of inner products among the basis elements.

Theorem 8. *Let $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$ be two $[n, k]$ linear codes with generator matrices $\mathbf{A}, \mathbf{B}$ respectively. Then*

$$\operatorname{rank}(\mathbf{A}\mathbf{B}^\top) = k - \dim_{\mathbb{F}_q} \mathscr{A} \cap \mathscr{B}^\perp.$$



(a) $h = \lfloor \sqrt{n} \rfloor$, without refinements

(b) $h = \lfloor \sqrt{n} \rfloor$, refinements with $z = 1$

(c) $h = \lfloor \sqrt{2n} - \frac{1}{2} \rfloor$, without refinements

(d) $h = \lfloor \sqrt{2n} - \frac{1}{2} \rfloor$, refinements with $z = 1$

Fig. 3: Success rates of our algorithm for several combinations of $(q, n, k = h)$. Each data point has been estimated by simulating the attack on 100 codes.

Proof. A vector $\mathbf{y}$ belongs to $\mathscr{A} \cap \mathscr{B}^{\perp}$ if and only if it can be written as $\mathbf{y} = \sum_{i=1}^{k} \alpha_i \mathbf{a}_i$ for some $\alpha_i \in \mathbb{F}_q$ and $\langle \mathbf{y}, \mathbf{b}_i \rangle = 0$ for all $i = 1, \ldots, k$. Let $\ker(\cdot)$ denote the right kernel. Given $\mathbf{y} = \sum_{i=1}^{k} \alpha_i \mathbf{a}_i \in \mathscr{A}$, we have that $(\alpha_1, \ldots, \alpha_k)^{\top} \in \ker(\mathbf{A}\mathbf{B}^{\top})$ if and only if

$$0 = \mathbf{G}\mathbf{G}'^{\top} \cdot \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix} = \begin{pmatrix} \langle \mathbf{y}, \mathbf{b}_1 \rangle \\ \vdots \\ \langle \mathbf{y}, \mathbf{b}_k \rangle \end{pmatrix},$$

i.e. if and only if $\mathbf{y} \in \mathscr{A} \cap \mathscr{B}^{\perp}$. It follows that

$$\dim_{\mathbb{F}_q} \ker(\mathbf{A}\mathbf{B}^{\top}) = \dim_{\mathbb{F}_q} \mathscr{A} \cap \mathscr{B}^{\perp}.$$

Since $\mathbf{A}\mathbf{B}^{\top}$ is a square matrix, we conclude that

$$\operatorname{rank}(\mathbf{A}\mathbf{B}^{\top}) = k - \dim_{\mathbb{F}_q} \ker(\mathbf{A}\mathbf{B}^{\top}) = k - \dim_{\mathbb{F}_q} \mathscr{A} \cap \mathscr{B}^{\perp}.$$

□

Remark 6. *Theorem 8 implies that, despite* $\mathbf{AB}^\top$ *depends on the basis elements (and even on their order), the rank is invariant and only depends on the codes* $\mathscr{A}, \mathscr{B}$*.*

By taking $\mathscr{A} = \mathscr{B}$, we immediately get

Corollary 1. *Let* $\mathscr{A} \subseteq \mathbb{F}_q^n$ *be an* $[n, k]$ *linear code with generator matrix* $\mathbf{A}$*. Then*

$$\operatorname{rank}(\mathbf{AA}^\top) = k - \dim_{\mathbb{F}_q} \mathcal{H}(\mathscr{A}).$$

We remark that $\mathbf{AA}^\top$ is a symmetric matrix.

Consider now a linear self-orthogonal code $\mathscr{C} \subseteq \mathbb{F}_q^n$ of dimension k such that $\binom{k+1}{2} < n$ and with generator matrix $\mathbf{G}$. For simplicity, we assume that the dimension of its square code attains the upper bound, i.e. $\dim_{\mathbb{F}_q} \mathscr{C}^2 = \binom{k+1}{2}$, and define its basis $\{\mathbf{c}_{i,j} \mid 1 \leqslant i \leqslant j \leqslant k\}$, where $\mathbf{c}_{i,j} = \mathbf{c}_i \star \mathbf{c}_j$.

The matrix $\mathbf{GG}^\top$ is the zero matrix, since $\mathscr{C}$ is self-orthogonal, and thus all the inner products $\langle \mathbf{c}_i, \mathbf{c}_j \rangle$ are null. We are rather interested in the matrix

$$\mathbf{M} \stackrel{\text{def}}{=} \mathbf{G}^{[2]} {\mathbf{G}^{[2]}}^\top \in \mathbb{F}_q^{d \times d}, \tag{4}$$

as $\mathscr{C}^2$ potentially breaks the self-orthogonality. Such a matrix has several properties depending on the field over which it is defined. We will use the following notation to denotes entries in $\mathbf{M}$:

$$\mathbf{M}_{(i_1,i_2),(j_1,j_2)} = \langle \mathbf{c}_{i_1,i_2}, \mathbf{c}_{j_1,j_2} \rangle = \langle \mathbf{c}_{i_1} \star c_{i_2}, \mathbf{c}_{j_1} \star c_{j_2} \rangle.$$

Proposition 4. *Let* $\mathscr{C} = \langle \mathbf{c}_1, \ldots, \mathbf{c}_k \rangle \subseteq \mathbb{F}_q^n$ *be an* $[n, k]$ *self-orthogonal code with* $\binom{k+1}{2} < n$*, and* $\mathbf{M} = \mathbf{G}^{[2]} {\mathbf{G}^{[2]}}^\top$*. Then:*

1. *for all* $i_1, i_2, j_1, j_2 \in [1, k]$, $\mathbf{M}_{(i_1,i_2),(j_1,j_2)} = \mathbf{M}_{(i_1,j_1),(i_2,j_2)}$*;*
2. *if* $q = 2^s$*, then all the diagonal entries are null, i.e. for all* $i, j \in [1, k]$*,* $\mathbf{M}_{(i,j),(i,j)} = 0$*;*
3. *if* $q = 2^s$*, then for all* $i, j \in [1, k]$*,* $\mathbf{M}_{(i,i),(j,j)} = 0$*;*
4. *if* $q = 2$ *or* $q = 3$*, then for all* $i, j \in [1, k]$*,* $\mathbf{M}_{(i,i),(i,j)} = 0$*.*

Proof. Property 1 is obtained directly from Lemma 1 by taking two different partitions with blocks of cardinality 2 of a set of 4 elements.

Property 2 is obtained from Lemma 1 in the following way:

$$\langle \mathbf{c}_{i,j}, \mathbf{c}_{i,j} \rangle = \langle \mathbf{c}_i \star \mathbf{c}_j, \mathbf{c}_i \star \mathbf{c}_j \rangle = \langle \mathbf{c}_i^2, \mathbf{c}_j^2 \rangle = \langle \mathbf{c}_i, \mathbf{c}_j \rangle^2 = 0.$$

Property 3 readily follows as a corollary by combining Property 1 with $i_1 = i_2 = i$, $j_1 = j_2 = j$ together with Property 2.

Finally, Property 4 results from Lemma 1 as

$$\langle \mathbf{c}_{i,i}, \mathbf{c}_{i,j} \rangle = \langle \mathbf{c}_i \star \mathbf{c}_i, \mathbf{c}_i \star \mathbf{c}_j \rangle = \langle \mathbf{c}_i^3, \mathbf{c}_j \rangle = \langle \mathbf{c}_i, \mathbf{c}_j \rangle = 0.$$

□

Due to Theorem 8, one can study the distribution of the hull dimension for the square of a self-orthogonal code from the rank of the corresponding matrix (4). We recall here some classic results from [Mac69], which count the number of symmetric matrices over a finite field of a given rank.

Proposition 5. *[Mac69, Theorems 2 & 3] Let $N(t,r)$ denote the number of symmetric matrices of size $t \times t$, rank $r \leqslant t$, with entries in a finite field $\mathbb{F}_q$. Let $N_0(t,r)$ denote the number of symmetric matrices of size $t \times t$, rank $r \leqslant t$, with entries in a finite field $\mathbb{F}_q$, $q = 2^m$, and 0 on the main diagonal. Then*

$$\begin{aligned} N(t,2s) &= \prod_{i=1}^{s} \tfrac{q^{2i}}{q^{2i}-1} \cdot \prod_{i=0}^{2s-1}(q^{t-i}-1), \\ N(t,2s+1) &= \prod_{i=1}^{s} \tfrac{q^{2i}}{q^{2i}-1} \cdot \prod_{i=0}^{2s}(q^{t-i}-1), \\ N_0(t,2s) &= \prod_{i=1}^{s} \tfrac{q^{2i-2}}{q^{2i}-1} \cdot \prod_{i=0}^{2s-1}(q^{t-i}-1), \\ N_0(t,2s+1) &= 0. \end{aligned}$$

Remark 7. *Proposition 5 already explains a very particular behavior, that is the constant parity of the hull dimension for fields of characteristic 2 (for codes $\mathscr{C}$ such that $\dim_{\mathbb{F}_q} \mathscr{C}^2 = \binom{k+1}{2}$). Indeed, similarly as what happens for skew-symmetric matrices over fields of characteristic $\neq 2$, symmetric matrices over $\mathbb{F}_{2^m}$ with 0's on the main diagonal can only have even rank. Hence, by Theorem 8, the hull dimension must have the same parity as $\binom{k+1}{2}$, i.e. it is odd if $k \equiv 1 \mod 4$ or $k \equiv 2 \mod 4$ and it is even if $k \equiv 3 \mod 4$ or $k \equiv 0 \mod 4$.*

Dividing the numbers $N(t,r)$ (resp. $N_0(t,r)$) by the total number of symmetric matrices, i.e. $q^{\binom{t+1}{2}}$ (resp. $q^{\binom{t}{2}}$), provides the relative frequency of a certain rank.

However, Proposition 4 shows that the matrix (4) corresponding to the square of a self-orthogonal code is not randomly distributed among all symmetric matrices. Instead, some very specific linear relations among its entries hold. Moreover, in the ternary and characteristic 2 cases, it is somewhat sparser. This means that such matrices depend on less than $\binom{\binom{k+1}{2}+1}{2}$ parameters. The next example illustrates this behavior.

Example 2. *We provide the generic matrix $\mathbf{M}$ for several parameters. Potentially different values in $\mathbb{F}_q$ are denoted by different symbols.*

– *Let $k = 3, q = 2$. Then*

$$\begin{array}{c} \begin{matrix} \mathbf{c}_{1,1} & \mathbf{c}_{1,2} & \mathbf{c}_{2,2} & \mathbf{c}_{1,3} & \mathbf{c}_{2,3} & \mathbf{c}_{3,3} \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & 0 & 0 \\ 0 & * & * & 0 & * & 0 \\ * & * & 0 & * & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \mathbf{c}_{1,1} \\ \mathbf{c}_{1,2} \\ \mathbf{c}_{2,2} \\ \mathbf{c}_{1,3} \\ \mathbf{c}_{2,3} \\ \mathbf{c}_{3,3} \end{matrix} \end{array}$$

– *Let $k = 3, q = 3$. Then*

$$\begin{array}{c} \begin{matrix} \mathbf{c}_{1,1} & \mathbf{c}_{1,2} & \mathbf{c}_{2,2} & \mathbf{c}_{1,3} & \mathbf{c}_{2,3} & \mathbf{c}_{3,3} \end{matrix} \\ \begin{pmatrix} 0 & 0 & + & 0 & \# & @ \\ 0 & + & 0 & \# & \% & * \\ + & 0 & 0 & \% & 0 & ? \\ 0 & \# & \% & @ & * & 0 \\ \# & \% & 0 & * & ? & 0 \\ @ & * & ? & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} \mathbf{c}_{1,1} \\ \mathbf{c}_{1,2} \\ \mathbf{c}_{2,2} \\ \mathbf{c}_{1,3} \\ \mathbf{c}_{2,3} \\ \mathbf{c}_{3,3} \end{matrix}$$

– *Let $k = 3, q = 2^m$, $m > 1$. Then*

$$\begin{array}{c} \begin{matrix} \mathbf{c}_{1,1} & \mathbf{c}_{1,2} & \mathbf{c}_{2,2} & \mathbf{c}_{1,3} & \mathbf{c}_{2,3} & \mathbf{c}_{3,3} \end{matrix} \\ \begin{pmatrix} 0 & * & 0 & - & \# & 0 \\ * & 0 & \$ & \# & \% & ? \\ 0 & \$ & 0 & \% & ; & 0 \\ - & \# & \% & 0 & ? & \& \\ \# & \% & ; & ? & 0 & ! \\ 0 & ? & 0 & \& & ! & 0 \end{pmatrix} \end{array} \begin{matrix} \mathbf{c}_{1,1} \\ \mathbf{c}_{1,2} \\ \mathbf{c}_{2,2} \\ \mathbf{c}_{1,3} \\ \mathbf{c}_{2,3} \\ \mathbf{c}_{3,3} \end{matrix}$$

– *Let $k = 4, q = 2$. Then*

$$\begin{array}{c} \begin{matrix} \mathbf{c}_{1,1} & \mathbf{c}_{1,2} & \mathbf{c}_{2,2} & \mathbf{c}_{1,3} & \mathbf{c}_{2,3} & \mathbf{c}_{3,3} & \mathbf{c}_{1,4} & \mathbf{c}_{2,4} & \mathbf{c}_{3,4} & \mathbf{c}_{4,4} \end{matrix} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & * & 0 & 0 & + & - & 0 \\ 0 & 0 & 0 & * & * & * & + & + & ! & + \\ 0 & 0 & 0 & * & 0 & 0 & + & 0 & ? & 0 \\ 0 & * & * & 0 & * & 0 & - & ! & - & - \\ * & * & 0 & * & 0 & 0 & ! & ? & ? & ? \\ 0 & * & 0 & 0 & 0 & 0 & - & ? & 0 & 0 \\ 0 & + & + & - & ! & - & 0 & + & - & 0 \\ + & + & 0 & ! & ? & ? & + & 0 & ? & 0 \\ - & ! & ? & - & ? & 0 & - & ? & 0 & 0 \\ 0 & + & 0 & - & ? & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} \mathbf{c}_{1,1} \\ \mathbf{c}_{1,2} \\ \mathbf{c}_{2,2} \\ \mathbf{c}_{1,3} \\ \mathbf{c}_{2,3} \\ \mathbf{c}_{3,3} \\ \mathbf{c}_{1,4} \\ \mathbf{c}_{2,4} \\ \mathbf{c}_{3,4} \\ \mathbf{c}_{4,4} \end{matrix}$$

Consider, for instance, the case $q = 2, k = 3$. The generic matrix depends only on one parameter and its rank is therefore either 0 (when $* = 0$) or 6 when (when $* = 1$). From our experiments, the hull distribution splits accordingly with approximately $1/2$ of the instances having hull dimension $6-6 = 0$ and the other half having hull dimension $6-0 = 6$. Proposition 5 remains, however, very useful to estimate the rank (and consequently the hull dimension) distribution for larger dimensions and/or field sizes. We tested this theoretical hull dimension distribution X against the experimental complementary rank distribution Y of matrices obtained from random self-orthogonal codes (over 10^4 samples), and we obtained a statistical distance $\frac{1}{2}\sum_{d \geqslant 0} |\mathbb{P}(X = d) - \mathbb{P}(Y = d)| < 0.01$ whenever

– $q = 2$ and $k \geqslant 8$;

– $q = 3$ and $k \geqslant 5$;
– $q = 4, 8, 9$ and $k \geqslant 4$;
– any k, otherwise.

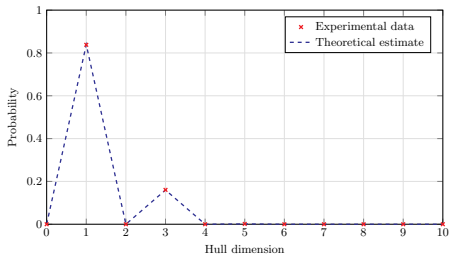


Fig. 4: Hull distribution for $q = 2, k = 9, n = 100$. The red plot shows the experimental distribution over 10000 tests while the blue plot is the estimation from Proposition 5.

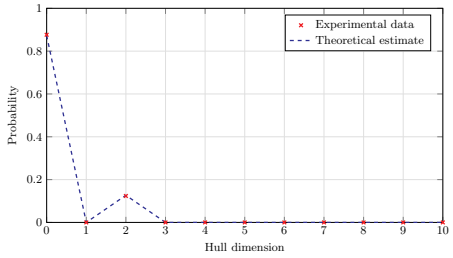


Fig. 5: Hull distribution for $q = 8, k = 4, n = 100$. The red plot shows the experimental distribution over 10000 tests while the blue plot is the estimation from Proposition 5.

The fact that the estimation improves as the dimension grows is not surprising, as in this case the number of parameters on which the matrix depends increases as well, making singular behaviors less probable. More precisely, we can upper-bound the number of parameters on which the generic matrix depends, based on the relations provided by Proposition 4.

Corollary 2. *Let* $\mathbf{M}$ *be the generic matrix constructed from an* $[n, k]_q$ *self-orthogonal code with* $\binom{k+1}{2} < n$*. Then:*

– *if* $q > 3$ *odd, then* $\mathbf{M}$ *depends on at most* $\binom{k+3}{4}$ *parameters;*
– *if* $q = 3$*, then* $\mathbf{M}$ *depends on at most* $\binom{k}{4} + 3\binom{k}{3} + \binom{k}{2}$ *parameters;*

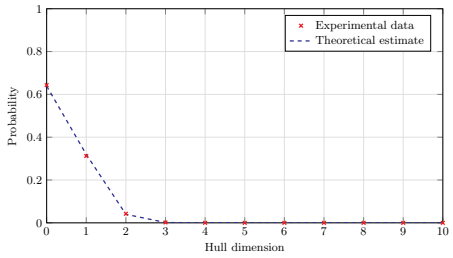


Fig. 6: Hull distribution for $q = 3, k = 6, n = 100$. The red plot shows the experimental distribution over 10000 tests while the blue plot is the estimation from Proposition 5.

- *if* $q = 2^m$, $m > 1$, *then* $\mathbf{M}$ *depends on at most* $\binom{k}{4} + 3\binom{k}{3} + 2\binom{k}{2}$ *parameters;*
- *if* $q = 2$, *then* $\mathbf{M}$ *depends on at most* $\binom{k}{4} + \binom{k}{3}$ *parameters.*

In all cases, the number of parameters is upper bounded by $\frac{k^4}{24}(1+o(1))$ *as* $k \to \infty$.

Proof. Property 1 of Proposition 4 associates entries $\mathbf{M}_{(i_1,i_2),(j_1,j_2)}$ to the multiset $\{i_1, i_2, j_1, j_2\}$ in such a way that if two entries have the same multiset then they are equal. The number of multisets of size 4 of the set $[1, k]$ is $\binom{k+3}{4}$.

If $q = 3$, Property 4 implies that multisets of the form $\{a, a, a, a\}$ or $\{a, a, a, b\}$, $b \neq a$ are associated with null entries. Hence, the number of parameters is at most

$$\binom{k+3}{4} - 2\binom{k}{2} - k = \binom{k}{4} + 3\binom{k}{3} + \binom{k}{2}.$$

If $q = 2^m$, $m > 1$, Property 2 implies that multisets of the form $\{a, a, b, b\}$ are associated with null entries. Hence, the number of parameters is at most

$$\binom{k+3}{4} - \binom{k}{2} - k = \binom{k}{4} + 3\binom{k}{3} + 2\binom{k}{2}.$$

Finally, if $q = 2$, Properties 2 and 4 imply that all the subsets with at at most 2 distinct elements are associated with null entries. Moreover, multisets with only three distinct elements are associated with entries with the same value if the corresponding sets are the same. Hence, the number of parameters coincides at most with the number of subsets of 3 or 4 elements:

$$\binom{k}{4} + \binom{k}{3}.$$

□

On the other hand, we experimentally verified for small values that the upper bounds on the number of parameters are tight. Hence, Corollary 2 estimates the matrix space dimension as a quartic power of k. For instance, when $k \in \Omega(\sqrt{n})$, the number of parameters is strictly smaller than the full symmetric matrix space

dimension (i.e. $\binom{n+1}{2}$ or $\binom{n}{2}$ depending on the field characteristic), but still linear with respect to it, since $\frac{k^4}{24}(1+o(1)) \in \Omega(n^2)$.

In the Hamming metric, the larger the rate of a code is the more likely its weight distribution is similar to the Bernoulli one. Here, analogously, showing that the matrix subspace is relatively large makes more plausible a good overlap with the distribution inferred by Proposition 5.

To summarize, the following heuristic on the hull dimension of the square of a self-orthogonal code, which is based on the previous explanations and experimental results, is sufficient to ensure that our algorithm works in polynomial time with high probability.

Heuristic 2. *Let $\mathscr{C} \subseteq \mathbb{F}_q^n$ be an $[n,k]$ self-orthogonal code such that $\dim_{\mathbb{F}_q} \mathscr{C}^2 = \binom{k+1}{2}$. For $k, n \to \infty$ and $k < \sqrt{2n}(1+o(1))$, the probability that $\mathcal{H}(\mathscr{C}^2)$ has dimension 0 can be estimated by*

- $\frac{N(\binom{k+1}{2},\binom{k+1}{2})}{q^{\binom{\binom{k+1}{2}+1}{2}}}$, *if q is odd;*
- $\frac{N_0(\binom{k+1}{2},\binom{k+1}{2})}{q^{\binom{\binom{k+1}{2}}{2}}}$, *if $q = 2^m$ and ($k \equiv 3 \mod 4$ or $k \equiv 0 \mod 4$).*

If $q = 2^m$ and ($k \equiv 1 \mod 4$ or $k \equiv 2 \mod 4$), the probability that $\mathcal{H}(\mathscr{C}^2)$ has dimension 1 can be estimated by

$$\frac{N_0(\binom{k+1}{2}, \binom{k+1}{2}-1)}{q^{\binom{\binom{k+1}{2}}{2}}},$$

where the values of N and N_0 are given in Proposition 5.

5 Cryptanalysis applications

In this section, we show how the theory developed in this paper can be used to devise a polynomial-time attack on the recommended instances of the Updatable Public-Key Encryption by Albrecht, Benčina, and Lai presented at Eurocrypt 2025 [ABL25].

5.1 The UPKE scheme [ABL25]

The paper [ABL25] presents an Updatable Public-Key Encryption (UPKE) scheme whose security as a PKE relies on the Learning With Errors (LWE) assumption [Reg09]. In a nutshell, a UPKE scheme consists of 5 algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{UpdPk}, \mathsf{UpdSk})$ such that

- $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a PKE scheme;
- any sender, including an adversary, can update another user's public key pk by running the algorithm UpdPk, obtaining a new public key pk' and an update token up;

– the receiver can use up to update their secret key sk to sk by running the algorithm UpdSk.

UPKE schemes have been introduced as a slight relaxation of forward-secure public-key encryption as a remedy against the poor efficiency and the requirement that parties stay synchronized in the latter construction. A UPKE scheme is considered secure if an exposed secret key sk' does not reveal the message encrypted with any previous public key, under the assumption that at least one update in the chain leading to sk' was not malicious. We refer the reader to [DKW21] for a more in-depth discussion of the topic and for the technical definitions of correctness and IND-CR-CPA security.

Previous post-quantum secure attempts at UPKE schemes are also based on LWE, but they suffer from a growing noise issue; thus, they only support a bounded number of updates. In contrast, the update mechanism of the scheme in [ABL25] has no practical upper bound on the number of updates. This is achieved through the use of the signed permutation equivalence problem (SPEP) assumption, defined below, in the random oracle model.

We first define $SP_n = S_n \ltimes \{\pm 1\}^{*n}$ as the group of signed permutation matrices, i.e. the matrices that can be written as $\mathbf{PD}$, where $\mathbf{P} \in S_n$ and $\mathbf{D}$ is a signature matrix, i.e. a diagonal matrix whose diagonal elements are 1 or -1, over $\mathbb{F}_q$.

Using signed permutation as isometries for the Hamming metric, we get the following problem. [4]

Problem 2. ***Signed Permutation Equivalence Problem (SPEP)***
Given two linear codes $\mathscr{A}, \mathscr{B} \subseteq \mathbb{F}_q^n$, decide if $\mathscr{A} \equiv_{SP_n} \mathscr{B}$, i.e., if there exists $\mu \in SP_n$ such that $\mathscr{B} = \mu(\mathscr{A})$. In other words, given two generator matrices $\mathbf{A}, \mathbf{B} \in \mathbb{F}_q^{k \times n}$, decide if there exist an invertible matrix $\mathbf{S} \in GL_k(\mathbb{F}_q)$ and a signed permutation matrix $\mathbf{M} \in SP_n$ such that $\mathbf{B} = \mathbf{SAM}$.

This variant of the code equivalence problem can be seen in a straightforward way as an intermediate version between PEP and LEP.

In [ABL25], the application of a uniformly random signed permutation to the key pair in the update mechanism does not provide any increase of noise, thus making the updating "free". More in detail, a uniformly random tall matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times k}$, representing a dual-Regev public key [GPV08] and where q is a prime, is seen as the generator matrix (in column shape) of a linear code over $\mathbb{Z}_q$. This code is sampled in such a way that the hull dimension h is sufficiently large to make algorithms like SSA [Sen00] and the reduction to GIP [BOST19] not efficient. In particular, h is such that $\min(q^h, n^h) \geqslant 2^\lambda$, for a given security parameter λ.

The recommended parameters are reported in Table 1; the security parameter is indicated with λ.

[4] The main reason to use signed permutation in the context of lattices is that they are also isometries for the Euclidean metric.

λ	$\log_2(q)$	n	k	h
128	13	7313	450	27
128	16	11000	550	26
192	18	20250	900	37
256	19	29688	1250	48

Table 1: Parameters of the instances from [ABL25]

5.2 Cryptanalysis

We observe that, in all the recommended instances, we always have $h \leqslant \sqrt{2n}$. This opens up the possibility of applying our attack.

As a little technicality, we have to deal with the fact that the scheme employs SPEP. However, this comes with very little effort. As showed in [DG23], it is easy to see that if $\mathscr{B} = \mu(\mathscr{A})$ with μ being a signed permutation, then $\mathcal{H}(\mathscr{B}) = \mu\big(\mathcal{H}(\mathscr{A})\big)$. Indeed, for μ a signed permutation, it holds that $\mathscr{B}^\perp = \mu(\mathscr{A}^\perp)$. To see this, let us express μ in matrix form as $\mathbf{PD}$ with $\mathbf{P}$ a permutation matrix and $\mathbf{D}$ a diagonal matrix with diagonal taking values in $\{\pm 1\}$. Let $\mathbf{G}$ denote a generator matrix for $\mathscr{A}$; then, a generator matrix for $\mathscr{B}$ is $\mathbf{G}' = \mathbf{GPD}$. For $\mathbf{H}$ a parity-check matrix for $\mathscr{A}$, we claim that a parity-check matrix for $\mathscr{B}$ is $\mathbf{H}' = \mathbf{HPD}$. Indeed, it holds that

$$\mathbf{G}'\mathbf{H}'^\top = \mathbf{GP}\underbrace{\mathbf{D}^2}_{\mathbf{I}_n}\mathbf{P}^\top\mathbf{H}^\top = \mathbf{GPP}^\top\mathbf{H}^\top = \mathbf{0}.$$

In the above equation, we used the fact that $\mathbf{D}^2 = \mathbf{I}_n$, since $(\pm 1)^2 = 1$.

Let $\mu \coloneqq (\mathbf{t}, \pi)$ with $\mathbf{t} \in \{\pm 1\}^n$. The permutation π can be found running our PEP attack on $\big(\mathcal{H}(\mathscr{A})\big)^2$ and $\big(\mathcal{H}(\mathscr{B})\big)^2$ since these two codes are only permutation equivalent and not signed permutation equivalent. Indeed, they would be mapped by the monomial represented by $\mathbf{PD}^2$ but, as we have already seen, $\mathbf{D}^2 = \mathbf{I}_n$. Using a PEP solver on $\big(\mathcal{H}(\mathscr{A})\big)^2$ and $\big(\mathcal{H}(\mathscr{B})\big)^2$, we can find π.

We now have to find $\mathbf{D}$. Let us denote by $\mathbf{P}$ be the matrix associated to π: to recover $\mathbf{D}$, it is enough to compute a parity-check matrix $\mathbf{H}'$ for $\mathscr{B}$ and then set up a linear system of the form

$$\mathbf{GPDH}'^\top = \mathbf{0}.$$

Let us indicate as $(t_1, \cdots, t_n)$ the elements in the diagonal of $\mathbf{D}$, and treat them as unknowns. Let $\widetilde{\mathbf{G}} = \mathbf{GP} = \{\widetilde{g}_{i,j}\}_{\substack{1 \leqslant i \leqslant k \\ 1 \leqslant j \leqslant n}}$; then, the above equation can be

rewritten as

$$\begin{gathered}\left\{\sum_{\ell=1}^{n} t_\ell \cdot \widetilde{g}_{i,\ell} h'_{j,\ell} = 0\right\}_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant n-k}} \\ \Downarrow \\ \left\{\widetilde{\mathbf{g}}_i \star \mathbf{h}'_j \cdot \mathbf{t}^\top = 0\right\}_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant n-k}}\end{gathered} \tag{5}$$

where $\widetilde{\mathbf{g}}_i$ is the i-th row of $\mathbf{G}$ and $\mathbf{h}'_j$ is the j-th row of $\mathbf{H}'$.

The vector $\mathbf{t}$ lies in right kernel of the matrix $\left\{\widetilde{\mathbf{g}}_i \star \mathbf{h}'_j\right\}_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant n-k}}$ which, with overwhelming probability, has dimension 1 and is hence generated by a unique vector which must be a scalar multiple of $\mathbf{t}$. In the unlikely case in which the kernel has dimension > 1, we enumerate all vectors in the kernel, searching for all non-null vectors containing either one or two values. Scaling these vectors yields the secret $\mathbf{t}$. Moreover, taking a random subset of slightly more than n out of the $k(n-k)$ $\widetilde{\mathbf{g}}_i \star \mathbf{h}'_j$ is sufficient in practice to solve the system with overwhelming probability.

The attack runs obviously in polynomial time, as all required operations are essentially basic linear algebra plus solving the PEP instance $\big(\mathcal{H}(\mathscr{A})\big)^2$, $\big(\mathcal{H}(\mathscr{B})\big)^2$ which, thanks to the analysis in this paper, is easy on average since the codes have trivial hulls with large probability. As a crude estimate, we can assess the complexity of the whole attack as $O\big(n^3 \log_2(q)\big)$, using $O\big(n^3 \log_2(q)\big)$ as the cost of the reduction to GIP.

Testing the attack We have tested the attack, using the reduction to GIP as a solver for PEP. To solve GIP, we relied on the basic Algorithm 2 given in Appendix A.

As a sanity check, we have implemented our attack on the instancies for 128 bits of security, recommended in [ABL25], using Sagemath.[5] For each parameter set, we have tried the attack on 100 randomly generated instances of Signed PEP. Notice that in [ABL25] the authors do not give the exact value for q, but only $\lceil \log_2(q) \rceil$. [6] Since q is a prime number, in our tests we set q as the largest prime smaller than corresponding power of 2. Analogous results hold for other prime numbers close to the same power of 2.

Since $n - \frac{h(h+1)}{2}$ is always much greater than $1 + \log_2(n)/\log_2(q)$, according to the analysis in Section 3 it is very unlikely that false positives or spurious solutions happen. For this reason, we ran the algorithm without refinements.

The attack has been tested on a standard laptop with a 13th Gen Intel(R) Core(TM) i7-1355U, with 16 GB RAM and running the Ubuntu 22.04.5 LTS operating system. For all tested instances, the squared codes had trivial hulls and

[5] The code is available: `https://github.com/psantini1992/SquareCodeEquiv`

[6] Essentially, q is chosen big enough to satisfy the correctness of the dual-Regev encryption.

our PEP solver on $\left(\mathcal{H}(\mathscr{A})\right)^2$ and $\left(\mathcal{H}(\mathscr{B})\right)^2$ has correctly retrieved the secret key. To reduce the size of $\{\widetilde{\mathbf{g}}_i \star \mathbf{h}'_j\}_{\substack{1\leqslant i\leqslant k\\ 1\leqslant j\leqslant n-k}}$, in our code we took random generator $\widetilde{\mathbf{G}}$ and parity-check $\mathbf{H}'$ matrices and selected a subset I for row-indexes i of cardinality slightly more than $\sqrt{nk/(n-k)}$. For each index i we selected a subset J_i for row-indexes j of cardinality slightly more than $\sqrt{n(n-k)/k}$ so that the total number of selected pairs (i,j) is slightly more than n. Alternatively, one can take $\widetilde{\mathbf{G}}$ and parity-check $\mathbf{H}'$ matrices in systematic form ad exploit the fact that the linear system is already partially reduced. In this case, however, one should pay attention to select, for any $1 \leqslant \ell \leqslant n$, at least one pair (i,j) such that $\widetilde{g}_{i,\ell} h'_{j,\ell} \neq 0$, so that the ℓ-th entry of $\mathbf{t}$ appears in the system. In all experiments, the vector $\mathbf{t}$ has been always correctly found.

The attack ran required, on average, 643 seconds for the first instance. We observe that the majority of time used by the attack is due to the linear algebra required for computing hulls, their squares and the graphs.

Countermeasure Notice that this weakness can be easily avoided by enlarging the hull dimension of the codes employed in the scheme from [ABL25]. A good solution may be that of choosing h to be larger than $\sqrt{2n}$. Values of h that are too close to $\sqrt{2n}$ shall be avoided since, using refinements on a sufficiently large number of positions, one may end up again with codes having a hull which is small enough to apply our attack. Indeed, if $h \approx \sqrt{2n} + z$, then one can apply our attack using, as refinements, puncturations on approximately z positions. This would lead to an overall cost of $O\left(\binom{n}{z} n^3 \log_2(q)\right)$. We recommend that z is set large enough, say, so that $\binom{n}{z} > 2^\lambda$.

Notice that, to be as conservative as possible, one may also rely on self-dual codes, for which $h = k$.

6 Conclusions and future work

In this article, we have identified new easy instances of the permutation and linear code equivalence problems. Such instances are vulnerable to the square-code construction, which has already been successful within the context of the cryptanalysis of McEliece-like cryptosystems, or more generally, to component-wise products and powers of codes. Previous applications aimed at distinguishing and recovering the secret representation of linear codes with a strong algebraic structure, based on the unusual behavior of the dimension of the square code.

On the other hand, the novelty of our attack consists of the fact that the target codes are potentially random. We have indeed seen how these techniques generally allow one to: 1) break the self-orthogonality while preserving the permutation equivalence, and 2) flatten the diagonal coefficients of the linear equivalence. Our results come with both experimental evidences and a theoretical explanation for the hull dimension distribution of the codes at hand, as well as for the false positives of our algorithms, which strengthen our claims about the success probability of the proposed method.

As a practical application, we have derived a probabilistic polynomial-time attack against the parameters of the underlying signed permutation equivalence problem used in the updating mechanism of the UPKE scheme Hollow-LWE [ABL25]. On the other hand, we have suggested a straightforward repair that consists of choosing codes with higher hull dimensions.

Our strategy exploits the reduction [BOST19] to the graph isomorphism problem. In addition to the previous contributions, we described and analyzed a simplified algorithm for solving the average-case graph isomorphism subroutine by means of efficiently computable invariants.

While our work on component-wise products differs from the previous literature in terms of the problem tackled and the randomness of the target codes, the limitations of this approach are essentially the same as those already identified for the cryptanalysis of McEliece-like schemes. In particular, the range of application is bound to degenerate parameters with sublinear dimension, and such a bound becomes increasingly restrictive as higher powers of codes are required. Our attack on LEP, for instance, does not threaten the security of LESS [BMPS20], as the codes there have rate $1/2$ and their powers trivially fill the ambient space with overwhelming probability. One could wonder if this research direction is thus intrinsically a dead end.

Recently, there have been advances regarding the distinguishing problem related to Goppa codes and the McEliece scheme, which can be interpreted as more powerful generalizations of the square-code approach. Computing the dimension of the square of a code $\mathscr{C}$ is indeed equivalent by duality to computing the dimension of the vector space of quadratic relations among the basis elements of the code itself.

In [CMT23], this space is associated with a space of quadratic forms and thus with a matrix code. Then, solving a MinRank instance corresponding to the matrix code provides a solution to the distinguishing problem. In other words, the idea is to use a metric different from Hamming—namely, the rank metric—for which some structure of these quadratic relations is invariant under a change of basis. In [Ran25], relations between these quadratic relations, called syzygies, are also computed. Then, such syzygies satisfy further relations, and syzygies of higher degree can be computed iteratively until an unexpected behavior in the Betti numbers (i.e., in the dimension of the syzygy modules) is observed.

In both cases, computation at a higher degree allows one to extract more information from the space of quadratic relations than just its dimension. The drawback is a substantial increase in computational complexity; nonetheless, these approaches have led to significant theoretical and asymptotic results. For instance, the Goppa code distinguisher in [Ran25] achieves, for the first time, a subexponential complexity in the error-correcting capability, which makes it asymptotically better than generic decoding algorithms.

It is then tempting to adjust these new algorithms to the code equivalence setting. While this task is not straightforward and goes well beyond the scope of this article, we believe that these techniques are potentially suitable for attacking the code equivalence problem, as they still provide valid invariants. For

instance, linearly equivalent codes have the same Betti numbers; see [Ran25, Lemma 3]. We thus leave the adaptation of "square-code generalizations" to the code equivalence problem as an interesting future direction.

Acknowledgements

The authors wish to thank Martin R. Albrecht, Benjamin Benčina, and Russell W. F. Lai, for the feedback and the valuable discussion about the implementation of the UPKE algorithm.

The first and third authors are partially supported by the Italian Ministry of University and Research (MUR) under the PRIN 2022 program with projects "Mathematical Primitives for Post Quantum Digital Signatures" (CUP I53D23006580001) and "Post quantum Identification and eNcryption primiTives: dEsign and Realization (POINTER)" (CUP I53D23003670006), by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan, funded by the European Union - Next Generation EU and by MUR under the Italian Fund for Applied Science (FISA 2022), Call for tender No. 1405 published on 13-09-2022 - project title "Quantum-safe cryptographic tools for the protection of national data and information technology assets" (QSAFEIT) - No. FISA 2022-00618 (CUP I33C24000520001), Grant Assignment Decree no. 15461 adopted on 02.08.2024. Michele Battagliola is also supported by the University of St.Gallen, Great Mind Fellowship, Project 103 I 638 for "Code-Based Cryptography for Post-Quantum Security (CoB-PQS)".

References

ABC+20. Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

ABL25. Martin Albrecht, Benjamin Benčina, and Russell WF Lai. Hollow LWE: A New Spin, Unbounded Updatable Encryption from LWE and PCE. In *EUROCRYPT 2025*. 2025.

Bab16. László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697, 2016.

BC14. Mikhail A Borodin and Ivan V Chizhov. Effective attack on the mceliece cryptosystem based on reed-muller codes. *Discrete Mathematics and Applications*, 24(5):273–280, 2014.

BCDD+24. Alessandro Budroni, Jesús-Javier Chi-Domínguez, Giuseppe D'Alconzo, Antonio J. Di Scala, and Mukul Kulkarni. Don't use it twice! solving

relaxed linear equivalence problems. In *Advances in Cryptology – ASIACRYPT 2024: 30th International Conference on the Theory and Application of Cryptology and Information Security, Kolkata, India, December 9–13, 2024, Proceedings. Part VIII*, page 35–65, Berlin, Heidelberg, 2024. Springer-Verlag.

Beu20. Ward Beullens. Not enough less: An improved algorithm for solving code equivalence problems over f q. In *International Conference on Selected Areas in Cryptography*, pages 387–403. Springer, 2020.

BHZ87. Ravi B. Boppana, Johan Hastad, and Stathis Zachos. Does co-np have short interactive proofs? *Information Processing Letters*, 25(2):127–132, 1987.

BM23. Jean-François Biasse and Giacomo Micheli. A search-to-decision reduction for the permutation code equivalence problem. In *2023 IEEE International Symposium on Information Theory (ISIT)*, pages 602–607. IEEE, 2023.

BMPS20. Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. Less is more: code-based signatures without syndromes. In *Progress in Cryptology-AFRICACRYPT 2020: 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20–22, 2020, Proceedings 12*, pages 45–65. Springer, 2020.

BMT23. Magali Bardet, Rocco Mora, and Jean-Pierre Tillich. Polynomial time key-recovery attack on high rate random alternant codes. *IEEE Transactions on Information Theory*, 70(6):4492–4511, 2023.

BN24. Alessandro Budroni and Andrea Natale. On the sample complexity of linear code equivalence for all code rates. Cryptology ePrint Archive, Paper 2024/1757, 2024.

BOST19. Magali Bardet, Ayoub Otmani, and Mohamed Saeed-Taha. Permutation code equivalence is not harder than graph isomorphism when hulls are trivial. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 2464–2468. IEEE, 2019.

CDAG20. Alain Couvreur, Thomas Debris-Alazard, and Philippe Gaborit. On the hardness of code equivalence problems in rank metric. *arXiv preprint arXiv:2011.04611*, 2020.

CGGU+14. Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. Distinguisher-based attacks on public-key cryptosystems using reed–solomon codes. *Designs, Codes and Cryptography*, 73:641–666, 2014.

CL21. Alain Couvreur and Matthieu Lequesne. On the security of subspace subcodes of reed–solomon codes for public key encryption. *IEEE Transactions on Information Theory*, 68(1):632–648, 2021.

CMCP17. Alain Couvreur, Irene Márquez-Corbella, and Ruud Pellikaan. Cryptanalysis of mceliece cryptosystem based on algebraic geometry codes and their subcodes. *IEEE Transactions on Information Theory*, 63(8):5404–5418, 2017.

CMT23. Alain Couvreur, Rocco Mora, and Jean-Pierre Tillich. A new approach based on quadratic forms to attack the mceliece cryptosystem. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–38. Springer, 2023.

CNP+23. Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika

Trimoska. Take your meds: digital signatures from matrix code equivalence. In *International Conference on Cryptology in Africa*, pages 28–52. Springer, 2023.

COT16. Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial time attack on wild mceliece over quadratic extensions. *IEEE Transactions on Information Theory*, 63(1):404–427, 2016.

CPS25. Tung Chou, Edoardo Persichetti, and Paolo Santini. On linear equivalence, canonical forms, and digital signatures. *Designs, Codes and Cryptography*, pages 1–43, 2025.

CPTZ25. Alain Couvreur, Rakhi Pratihar, Nihan Tanısalı, and Ilaria Zappatore. On the structure of the schur squares of twisted generalized reed-solomon codes and application to cryptanalysis. In *International Conference on Post-Quantum Cryptography*, pages 3–34. Springer, 2025.

CSV25. Mahdi Cheraghchi, Nikhil Shagrithaya, and Alexandra Veliche. Reductions between code equivalence problems. Cryptology ePrint Archive, Paper 2025/208, 2025.

DBB18. Vlad Drăgoi, Valeriu Beiu, and Dominic Bucerzan. Vulnerabilities of the mceliece variants based on polar codes. In *International Conference on Security for Information Technology and Communications*, pages 376–390. Springer, 2018.

DG23. Léo Ducas and Shane Gibbons. Hull attacks on the lattice isomorphism problem. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 177–204, Cham, 2023. Springer Nature Switzerland.

DKW21. Yevgeniy Dodis, Harish Karthikeyan, and Daniel Wichs. Updatable public key encryption in the standard model. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part III 19*, pages 254–285. Springer, 2021.

DvW22. Léo Ducas and Wessel van Woerden. On the lattice isomorphism problem, quadratic forms, remarkable lattices, and cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2022.

FGUO+13. Jean-Charles Faugere, Valérie Gauthier-Umana, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high-rate mceliece cryptosystems. *IEEE Transactions on Information Theory*, 59(10):6830–6844, 2013.

GPV08. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206, 2008.

HP10. W Cary Huffman and Vera Pless. *Fundamentals of error-correcting codes*. Cambridge university press, 2010.

Leo82. Jeffrey Leon. Computing automorphism groups of error-correcting codes. *IEEE Transactions on Information Theory*, 28(3):496–511, 1982.

Mac69. Jessie MacWilliams. Orthogonal matrices over finite fields. *The American Mathematical Monthly*, 76(2):152–164, 1969.

Mat79. Rudolf Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.

McE78. Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244(1978):114–116, 1978.

MP14. Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

MS88. Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes*, volume 16 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, Netherlands, 1st edition, 1988.

MT23. Rocco Mora and Jean-Pierre Tillich. On the dimension and structure of the square of the dual of a goppa code. *Designs, Codes and Cryptography*, 91(4):1351–1372, 2023.

Now25. Julian Nowakowski. An improved algorithm for code equivalence. In *International Conference on Post-Quantum Cryptography*, pages 71–103. Springer, 2025.

PR97. Erez Petrank and Ron M Roth. Is code equivalence easy to decide? *IEEE Transactions on Information Theory*, 43(5):1602–1604, 1997.

Ran15. Hugues Randriambololona. On products and powers of linear codes under componentwise multiplication. *Algorithmic arithmetic, geometry, and coding theory*, 637:3–78, 2015.

Ran25. Hugues Randriambololona. The syzygy distinguisher. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 324–354. Springer, 2025.

Reg09. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

RS09. L.B. Richmond and Jeffrey Shallit. Counting abelian squares. *The Electronic Journal of Combinatorics [electronic only]*, 16(1):Research Paper R72, 9 p.–Research Paper R72, 9 p., 2009.

RST24. Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. Hardness estimates of the code equivalence problem in the rank metric. *Designs, Codes and Cryptography*, pages 1–30, 2024.

Sae17. Mohamed Ahmed Saeed. *Algebraic approach for code equivalence*. PhD thesis, Normandie Université; University of Khartoum, 2017.

Sch88. Uwe Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

Sen97. Nicolas Sendrier. On the dimension of the hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293, 1997.

Sen00. Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.

SS13. Nicolas Sendrier and Dimitris E Simos. The hardness of code equivalence over and its application to code-based cryptography. In *International Workshop on Post-Quantum Cryptography*, pages 203–216. Springer, 2013.

Tor04. Jacobo Torán. On the hardness of graph isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004.

Wie10. Christian Wieschebrink. Cryptanalysis of the niederreiter public key scheme based on grs subcodes. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3*, pages 61–72. Springer, 2010.

A A basic solver for the Graph Isomorphism Problem

The algorithm we consider in this paper to solve the search version of GIP is given in Algorithm 2. The algorithm uses, as a subroutine, a function Count that

takes a tuple of n finite field elements $(a_1, \cdots, a_n)$ and returns a length-q vector of integers whose i-th entry corresponds to the number of occurrences of the i-th finite field element. This can be implemented in time $O\big(n \log_2(q)\big)$.

Algorithm 2: Simple algorithm to solve search GIP

Data: finite field size q, code length n
Input: adjacency matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{F}_q^{n \times n}$
Output: permutation $\mathbf{P} \in S_n$, if it exists, such that $\mathbf{P}^\top \mathbf{X} \mathbf{P} = \mathbf{Y}$, or failure $\perp$ otherwise

Set $\mathbf{P} = \mathbf{0}$; `// null matrix`

`// Compute list with multiset entries for rows of` $\mathbf{X}$
Initialize $\mathcal{L} = \varnothing$;
for $i = 1, \cdots, n$ **do**
    Compute $\mathbf{a}^{(i)} = \mathsf{Count}(x_{i,1}, \cdots, x_{i,n})$;
    Update $\mathcal{L} \leftarrow \mathcal{L} \cup \{(i, \mathbf{a}^{(i)})\}$;

`// Compute multiset entries for rows of` $\mathbf{Y}$`, search for collisions`
for $j = 1, \cdots, n$ **do**
    Compute $\mathbf{b}^{(j)} = \mathsf{Count}(y_{j,1} \cdots, y_{j,n})$;
    Find $I \subseteq \{1, \cdots, n\}$ such that, for each $i \in I$, $(i, \mathbf{a}^{(i)}) \in \mathcal{L}$ and $\mathbf{a}^{(i)} = \mathbf{b}^{(j)}$;
    if $|I| \neq 1$ **then**
        return $\perp$;
    else
        Let $I = \{i\}$; set $p_{i,j} = 1$;

`// Check if` $\mathbf{P}$ `is a permutation matrix, report failure otherwise`
if $\mathbf{P}$ is not a permutation matrix **then**
    return $\perp$;
return $\mathbf{P}$

The algorithm receives as input adjacency matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{F}_q^n$ for two weighted, undirected graphs and aims to find, if it exists, a permutation $\mathbf{P} \in S_n$ so that $\mathbf{P}^\top \mathbf{X} \mathbf{P} = \mathbf{Y}$. This is done by matching rows according to the multiset formed by their entries: if for row j of $\mathbf{Y}$ there is a unique row in $\mathbf{X}$, say, row i, having the same entry multiset, then the permutation must send i to j. This matching among rows is performed for all rows of $\mathbf{Y}$: if it is successful for every row, then the algorithm found the permutation. If, for a row in $\mathbf{Y}$, there is more than one matching rows in $\mathbf{X}$, then the algorithm returns a failure.

Observe that there exist more precise algorithms to solve GIP. Yet, as we motivate next, this algorithm is expected to succeed with large probability when both q and n are large enough.

We first recall a result which will be instrumental in our analysis.

Lemma 2. *[RS09] Let* $(\mathbf{a}, \mathbf{b}) \xleftarrow{\$} \mathbb{F}_q^n \times \mathbb{F}_q^n$. *If* $q = 2$, *then* $\Pr[\mathsf{Count}(\mathbf{a}) = \mathsf{Count}(\mathbf{b})] = \binom{2n}{n} 4^{-n}$ *and, for growing* n, $\Pr[\mathsf{Count}(\mathbf{a}) = \mathsf{Count}(\mathbf{b})] \sim f_{2,n} = \frac{1}{\sqrt{\pi n}}$.

If $q \geqslant 3$, *then* $\Pr[\mathsf{Count}(\mathbf{a}) = \mathsf{Count}(\mathbf{b})] = \binom{2n}{n} 2^{-n} \leqslant \frac{1}{q^n} \cdot f_{q,n}$, *where*

$$f_{q,n} = \begin{cases} n! & \text{if } q \geqslant n, \\ \frac{n!}{(v!)^{q(v+1)-n}\left((v+1)!\right)^{n-qv}} & \text{if } q < n, \end{cases}$$

with $v = \lfloor n/q \rfloor$. *For constant* q *and growing* n, *then* $\Pr[\mathsf{Count}(\mathbf{a}) = \mathsf{Count}(\mathbf{b})]$ *is upper bounded by* $\frac{q^{\frac{q}{2}}}{(4\pi n)^{\frac{q}{2}-\frac{1}{2}}}$.

Remark 8. *For* $q = 2$ *we have been able to derive an exact formula as a collision in the computation of* Count *means that the vectors* $\mathbf{a}$ *and* $\mathbf{b}$ *have the same Hamming weight. For* $q \geqslant 3$, *having the same Hamming weight is a necessary but not sufficient condition and one has to consider also how the number of non-null entries distribute: this makes computations rather cumbersome, especially if one wants to derive a closed form formula. This is the reason why we have relied for a much easier-to-handle upper bound.*

Remark 9. *For constant* q, *the quantity* $\frac{f_{q,n}}{q^n}$ *is in* $O\left(n^{\frac{1-q}{2}}\right)$.

We now prove that, for any prime $q \geqslant 5$ and growing n, the probability that a random $n \times n$ matrix does not have rows having the same entry multiset is in $O(1)$.

Lemma 3. *Let* $\mathbf{X} \xleftarrow{\$} \mathbb{F}_q^{n \times n}$; *then, for constant* $q \geqslant 5$ *and growing* n, *the probability that all rows of* $\mathbf{X}$ *have distinct entries multiset is in* $O(1)$. *For* $q < 5$, *the probability is in* $o(1)$.

Proof. We know that, for any row of $\mathbf{X}$, the number of vectors having the same entries multiset is not greater than $f_{q,n}$. Then, the probability we are searching for is at least

$$\prod_{i=1}^{n-1} 1 - \frac{i \cdot f_{q,n}}{q^n} > \left(1 - \frac{(n-1) \cdot f_{q,n}}{q^n}\right)^{n-1} > \left(1 - \frac{n \cdot f_{q,n}}{q^n}\right)^n.$$

We now bound $\frac{f_{q,n}}{q^n}$ with $\frac{q^{q/2}(4\pi)^{q/2-1/2}}{n^{q/2-1/2}}$ and get

$$\begin{aligned} \left(1 - \frac{n \cdot f_{q,n}}{q^n}\right)^n &> \left(1 - \frac{n \cdot q^{q/2}(4\pi)^{q/2-1/2}}{n^{q/2-1/2}}\right)^n \\ &= \left(1 - \frac{q^{q/2}(4\pi)^{q/2-1/2}}{n^{q/2-3/2}}\right)^n \\ &\sim e^{-(4\pi)^{(1-q)/2} \cdot q^{\frac{q}{2}} \cdot n^{(5-q)/2}}. \end{aligned}$$

For $q = 5$, the exponent is a constant in n. For any $q \geqslant 7$, the exponent grows as the product between a negative constant, given by $-(4\pi)^{(1-q)/2} \cdot q^{\frac{q}{2}}$, and a polynomial in n that decays as fast as $1/\sqrt{n}$. Hence, for growing n, the exponent is in $o(1)$ and the above bound approaches 1. □ □

We can now use the above Lemma to estimate the probability that, for a random adjacency matrix $\mathbf{X} \in \mathbb{F}_q^{n \times n}$, all rows have distinct entries multisets: when this condition is met, Algorithm 2 can successfully solve search GIP. Observe that this requires to heuristically assume that the symmetry of the adjacency matrix does not influence the probability. We have verified this heuristic with numerical simulations on random symmetric matrices, for several values of q and growing n; for each parameter set, we have measured the probability that at least two rows have the same entries multiset. The results are shown in Figure 7. As we can see, the figure confirms the heuristic: for $q = 2$ or $q = 3$, the probability decreases as n increases, for $q = 5$ the probability stays approximately constant while, for $q \geqslant 7$, it approaches 1.

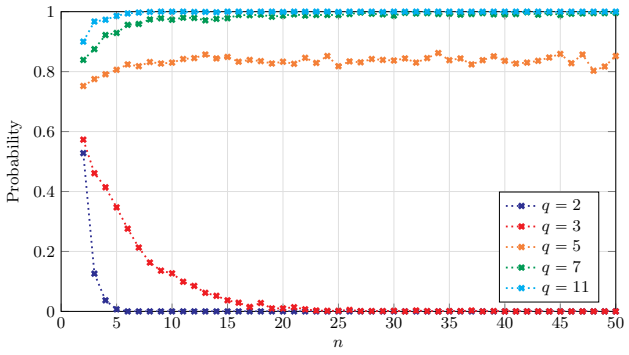


Fig. 7: Simulated probability of having at least two rows with the same multiset entries, for random $n \times n$ symmetric matrices over $\mathbb{F}_q$. For each pair (q, n), we have generated 1000 matrices.